UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Uma abordagem para auxiliar a correção de erros de programadores iniciantes

Dissertação de Mestrado

Galileu Santos de Jesus

São Cristóvão – Sergipe

2018

UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Galileu Santos de Jesus

Uma abordagem para auxiliar a correção de erros de programadores iniciantes

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Sergipe como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Orientador(a): Prof. Dr. Alberto Costa Neto

São Cristóvão – Sergipe

2018

Galileu Santos de Jesus
Uma abordagem para auxiliar a correção de erros de programadores iniciantes/ Galileu Santos de Jesus. – São Cristóvão – Sergipe, 2018-
156 p. : il.

Orientador: Prof. Dr. Alberto Costa Neto

Dissertação de Mestrado – UNIVERSIDADE FEDERAL DE SERGIPE
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO, 2018.

1. Juiz *on-line*. 2. Dicas de programação para iniciantes. 3. Ensino de programação. 4. Erros de sintaxe ou compilação em python. 5. Programação de computadores.

CDU 004.416.6

Galileu Santos de Jesus

Uma abordagem para auxiliar a correção de erros de programadores iniciantes

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Sergipe como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Trabalho aprovado. São Cristóvão – Sergipe, 24 de Janeiro de 2018:

Prof. Dr. Alberto Costa Neto
Orientador

Prof. Dr. Rodrigo de Barros Paes, Membro
Universidade Federal de Alagoas (UFAL)

Prof. Dra. Leila Maciel de Almeida e Silva, Membro
Universidade Federal de Sergipe (UFS)

São Cristóvão – Sergipe
2018

I dedicate this thesis to all my family, friends and
professors who gave me the necessary support to get here. Thank you so much for everything!

Agradecimentos

Aos meus orientadores de graduação, que sempre se dispuseram a idealizar projetos comigo no decorrer do curso, foram eles: Leila Almeida Silva, Alberto Costa Neto, Edward Davi Moreno e Beatriz Trinchão Andrade.

Agradeço muito a Leila por me acompanhar e orientar desde os meus 14 anos de idade. Hoje sou um profissional porque enxerguei através de você novos horizontes e novos caminhos. Você foi e é muito importante em minha vida! Muito obrigado.

Ao meu orientador, por estar comigo a mais de 3 anos. Obrigado não só por orientar todos os trabalhos, mas por inspirar, motivar, pelas oportunidades, por acreditar em mim e por ser quem você é! Não foi ao acaso que te escolhi e você me aceitou. Muito obrigado.

Agradeço a todos os meus professores, desde o ensino básico ao superior. Saibam que vocês foram de extrema importância para meu crescimento profissional e pessoal. Levo um pedacinho de cada um em minha vida, pois as sementes foram plantadas e regadas.

Aos meus familiares que sempre estiveram ao meu lado, em especial minhas irmãs: Daniela, Deise, Jakeline e Vaninha.

Ao meu pai e à minha mãe por serem minha inspiração. Saibam que os amo muito. É e foi por vocês! Palavras não descrevem a sensação de tê-los em minha vida! Obrigado por tudo!

GRATIDÃO!

A estrada vai além do que se vê...

(Los Hermanos)

Viver não cabe no lattes!

Resumo

A utilização de ambientes virtuais de aprendizagem integrados a outras ferramentas, como juízes *on-line*, surgem como uma possibilidade de amenizar a carência de práticas laboratoriais em cursos presenciais, além de poder apoiar atividades práticas em cursos semipresenciais, a distância e em MOOC's, assim como dar suporte aos docentes, possibilitando um melhor acompanhamento de rendimento individual. Porém, nem sempre os juízes *on-line* fornecem o *feedback* apropriado ou entendível pelo aluno, isto é, normalmente não fornecem dicas ao aluno de como melhorar ou alcançar uma solução válida. Ao observar turmas iniciais, verifica-se que frequentemente são apresentadas as mesmas dicas, já que os alunos costumam errar muito em um mesmo ponto ou por uma mesma razão. Este trabalho apresenta uma proposta para apoiar o ensino-aprendizagem de programação de computadores, aprimorando o juiz *on-line* The Huxley através da capacidade de produzir mensagens de *feedback* que sejam facilmente compreendidas pelos aprendizes de disciplinas iniciais de programação, norteando-os sobre os erros de sintaxe apresentados ao realizar uma submissão ao juiz *on-line*. Também foi feito um estudo de caso com turmas de graduação para avaliar esta abordagem, através de um experimento controlado, assim como sua análise com testes estatísticos para confirmação de hipótese, onde o estudo concluiu que a abordagem aumentou a capacidade de corrigir erros, além de guiá-los mais enfaticamente, principalmente entre alunos com baixo domínio da língua inglesa e que lograram êxito na disciplina inicial de programação.

Palavras-chave: juiz *on-line*, dicas de programação para iniciantes, ensino de programação, erros de sintaxe ou erros de execução em python, programação de computadores.

Abstract

The integration of virtual learning environments integrated with other tools, such as online judges, appears as a possibility to compensate the lack of laboratory practices in face-to-face courses, to support practical activities in semi-distance, distance and MOOC's courses, as well as supporting teachers, enabling a better monitoring of individual. However, online judges do not always provide an appropriate or understandable feedback to the student, they usually do not support the student with hints based on how to improve or achieve a valid solution. When observing groups of beginners, the same tips are presented frequently, since the students usually fail in the same point or by the same reason. This dissertation presents a proposal to support teaching-learning of computer programming, improving the online judge The Huxley by including feedback messages that are easily understood by the learners of the initial programming disciplines and guiding them through the syntax errors presented when performing a submission to the online judge. In order to evaluate this approach, a case study with undergraduate classes was also conducted. A controlled experiment, including an analysis with statistical tests confirms the hypothesis, that the approach increased the ability to correct errors, especially among students with low English proficiency that have succeeded in the initial programming discipline.

Keywords: online judge, tips for programming beginners, teaching programming, syntax errors or runtime errors in python, computer programming.

Lista de ilustrações

Lista de tabelas

Lista de códigos

Lista de abreviaturas e siglas

API	Application Programming Interface
IDE	Integrated Development Environment
UFS	Universidade Federal de Sergipe
TH	The Huxley
AVA	Ambiente Virtual de Aprendizagem
MOODLE	Modular Object Distance Learning
EOF	End Of File
RF	Requisito Funcional
EOL	End Of Line
MOOC	Massive Open Online Course

Sumário

1 Introdução

Neste capítulo é apresentada inicialmente na Seção 1.1 uma breve introdução contendo a contextualização acerca do tema de pesquisa. Em seguida, a Seção 1.2 apresenta a justificativa do trabalho. Na Seção 1.3, os objetivos do trabalho são discutidos. A Seção 1.4, apresenta a hipótese que se pretende provar com a realização deste trabalho. As principais contribuições do trabalho durante o seu desenvolvimento são descritas na Seção 1.5. Por fim, na Seção 1.6, a estrutura da organização da dissertação é descrita.

1.1 Contextualização

A computação está cada vez mais influenciando diversas áreas de ensino, segundo Júnior e Rapkiewicz (2004) ao se propor uma solução utilizando computação, obtêm-se soluções para um conjunto de problemas e processos para diversas classes de pessoas, de organizações e ambientes. Com isso, a busca por profissionais que sejam capazes de lidar com novas tecnologias aumenta constantemente.

Em turmas de ensino de programação de computadores, é fundamental que haja práticas laboratoriais. Porém, devido ao grande número de alunos, muitas vezes isto não ocorre. A fim de sanar o problema do acompanhamento das turmas e aulas práticas, pesquisas são desenvolvidas para tentar amenizar esta demanda (JÚNIOR; RAPKIEWICZ, 2004; MARCOLINO; BARBOSA, 2015).

Os Ambientes Virtuais de Aprendizagem - AVA - surgiram como uma forma de assessorar o ensino de aprendizagem em diversas áreas. Segundo Cardoso (2010), um AVA consiste em um sistema de computador em rede ou na Internet que possui um conjunto de ferramentas e recursos tecnológicos integrados, os quais permitem mediar o processo ensino-aprendizagem, seja ele presencial, a distância ou semipresencial, tendo como objetivo a construção de estratégias pedagógicas e montagem de cursos *on-line* ou para apoio em cursos presenciais. Possibilita

o armazenamento, o gerenciamento, a atualização/edição e o fluxo de recursos e conteúdos educacionais. Esta ferramenta de ensino vem tomando espaço cada vez maior no ramo de educação a distância.

Ainda segundo Cardoso (2010), os AVAs ganham uma importância crescente nos cursos presenciais ou a distância, sendo usados em diferentes instituições de ensino, tanto públicas quanto privadas, abrindo espaço para o despertar de diversas soluções similares, com o intuito de atender as necessidades de cada área de ensino. O principal objetivo destes ambientes é promover o relacionamento aluno-aluno para a construção e difusão do conhecimento de maneira virtual e colaborativa sob mediação do professor, pressupondo-se a autonomia intelectual dos discentes. Os AVA's se tornaram um instrumento de grande potencial nos novos modelos pedagógicos que envolvem a participação ativa dos alunos, sem excluir a presença dos professores.

Segundo Pereira, Schmitt e Dias (2007), o processo de ensino-aprendizagem tem potencial para tornar-se mais ativo, dinâmico e personalizado por meio de Ambientes Virtuais de Aprendizagem, visto que estas mídias utilizam o ciberespaço para promover a interação e a colaboração a distância entre os atores do processo e a interatividade com o conteúdo a ser aprendido.

Existem ferramentas específicas que auxiliam professores e alunos nas disciplinas de programação de computadores, conhecidas como juiz *on-line*. Um juiz *on-line* é um repositório web composto de problemas de programação disponíveis para os usuários submeterem suas respostas em forma de código-fonte em uma determinada linguagem de programação. Seu objetivo principal é avaliar a submissão, gerando respostas como: certo, saída errada, saída mal formatada, erro de compilação, erro em tempo de execução, entre outros. Alguns desses sistemas possuem integração com algumas funcionalidade dos Ambientes de Aprendizagem, disponibilizando fórum de discussões de estratégias para solucionar determinado problema, acompanhamento de turmas e rendimento individual dos alunos. Além disso, os alunos possuem a liberdade para escolher em qual linguagem de programação submeterá a sua resposta (KURNIA; LIM; CHEANG, 2001; SANTOS; RIBEIRO, 2011; CAMPOS; FERREIRA, 2004).

Em exercícios de programação é comum haver mais de uma solução para um determinado problema. Portanto, torna-se difícil para o educador avaliar individualmente todas as soluções de um determinado exercício e pontuar de acordo com a quantidade de acertos ou erros dos estudantes (MOREIRA; FAVERO, 2009).

Para ensinar programação, é extremamente importante que o aluno treine a implementação dos algoritmos (sequência de passos para realizar uma computação) em uma linguagem de programação. É natural que o aluno tenha vários problemas durante a implementação dos programas, como: a configuração inicial do ambiente de programação, erros de sintaxe dos programas e falhas na lógica do algoritmo implementado. É por isso que, para um bom aprendizado, é preciso que o aluno tenha uma resposta rápida sobre a corretude de seus programas, de forma que possa identificar suas falhas e gradativamente aumentar a complexidade dos problemas resolvidos

(WEBER; BRUSILOVSKY; STEINLE, 2014; PRIOR, 2003).

Segundo Weber, Brusilovsky e Steinle (2014), a tarefa de aprender a programar inclui:

- Adquirir habilidade em resolução de problemas, que envolve a identificação de metas e construção de planos de programação;
- Aprender a sintaxe de uma linguagem de programação;
- Aprender sobre lógica de programação, ou seja, entender o comportamento do computador na execução de um programa, o que inclui compreender a semântica da linguagem de programação;
- Utilização de um ambiente de programação;
- Realização de testes e depuração de programas.

Ensinar programação de computadores, é uma tarefa complexa (WEBER; BRUSILOVSKY; STEINLE, 2014), que exige não somente do professor, pois é necessário que o aluno possua uma certa habilidade. Há uma forte indicação de que o conhecimento anterior sobre a resolução de problemas dentro do contexto anterior à computação é um pré-requisito importante no aprendizado de linguagens de programação (LEMOS; BARROS; LOPES, 2003). No caso dos alunos, estes costumam ter muita dificuldade em aplicar suas habilidades prévias, criando fonte de medo e frustração, segundo (JÚNIOR; RAPKIEWICZ, 2004).

Apesar da emergência de ferramentas que auxiliem o aprendizado, tais como AVA's e juízes *on-line*, que facilitam o processo de ensino de programação de computadores, estas ferramentas ainda precisam evoluir bastante em alguns aspectos. Este trabalho aborda as dificuldades dos alunos em resolver problemas de programação para disciplinas iniciais, que muitas vezes se tornam objeto de medo e frustração, pois as ferramentas frequentemente não fornecem um *feedback* de forma clara e concisa. O público alvo são alunos que estão aprendendo a programar, os quais estão desfrutando do primeiro contato com uma linguagem de programação e por isso estão ainda aprendendo a utilizar conceitos introdutórios como estruturas condicionais, estruturas de laços de repetição, declaração de funções, uso de funções e comandos específicos, declaração de variáveis e outros. O objetivo é fornecer dicas em Português que orientem melhor o aluno a resolver um determinado erro de sintaxe.

Para exemplificar esta problemática, a Figura 1 ilustra uma mensagem de *feedback* fornecida ao aluno, ao realizar uma submissão utilizando a linguagem Python, no sistema de juiz *on-line* The Huxley. A transcrição da saída produzida pelo interpretador da linguagem Python exibe que o aluno tentou utilizar o comando de entrada ou função *input()* da linguagem supracitada, porém digitou incorretamente: "*iput()*", então foi exibida a seguinte mensagem: "*NameError: name 'iput' is not defined*- linha 4 e linha 25, sinalizando que o nome *iput()* não foi

definido. Porém, foram exibidos diversos erros pertencentes ao sistema de exceção, encontrados nas linhas 6 a 19, não trazendo nenhuma informação útil que possa sanar o problema.

```
1  Traceback (most recent call last):
2    File "/3.py", line 3, in <module>
3      n=int(iput())
4  NameError: name 'iput' is not defined
5  Error in sys.excepthook:
6  Traceback (most recent call last):
7    File "/usr/lib/python3/dist-packages/apport_python_hook.py", line 63, in apport_excepthook
8      from apport.fileutils import likely_packaged, get_recent_crashes
9    File "/usr/lib/python3/dist-packages/apport/__init__.py", line 5, in <module>
10     from apport.report import Report
11   File "/usr/lib/python3/dist-packages/apport/report.py", line 30, in <module>
12     import apport.fileutils
13   File "/usr/lib/python3/dist-packages/apport/fileutils.py", line 23, in <module>
14     from apport.packaging_impl import impl as packaging
15   File "/usr/lib/python3/dist-packages/apport/packaging_impl.py", line 20, in <module>
16     import apt
17   File "/usr/lib/python3/dist-packages/apt/__init__.py", line 34, in <module>
18     apt_pkg.init_config()
19 SystemError: E:Unable to read /etc/apt/apt.conf.d/ - opendir (13: Permission denied)
20
21 Original exception was:
22 Traceback (most recent call last):
23   File "/3.py", line 3, in <module>
24     n=int(iput())
25 NameError: name 'iput' is not defined
```

Figura 1 – Exemplo de erro de sintaxe fornecido pelo The Huxley para uma submissão realizada na linguagem Python.

Segundo Miyadera, Huang e Yokoyama (2000) uma das maiores dificuldades de um estudante de programação é entender cada passo da execução do programa. Assim como entender e aprender a gramática de uma linguagem de programação, consertar erros de sintaxe em um programa, desenvolver novos algoritmos, escrever um novo programa, depurar e consertar erros em um programa, em ordem crescente de dificuldade.

Para estudantes iniciantes em programação, a mensagem supracitada pode ser muitas vezes fonte de medo e frustração, visto que é uma mensagem de difícil compreensão, frequentemente coibindo o aprendiz em progredir na aprendizagem em sala de aula. Além disso, para entender a mensagem é requerido um conhecimento mínimo não só da língua inglesa mas também de erros referentes ao processo de compilação ou interpretação.

Apesar de os juízes *on-line* ajudarem bastante no processo de ensino aprendizagem das disciplinas iniciais de programação, muitas das vezes são fornecidas mensagens de difícil compreensão para iniciantes. Alguns casos podem demandar grande esforço, não só para o aluno, mas também para o professor. Na hora de solucionar um determinado erro de compilação/execução, que são de comum ocorrência no início da aprendizagem, seria interessante que as possíveis causas fossem apontadas através de algumas dicas geradas automaticamente.

Todo o material utilizado neste trabalho, encontra-se disponível em uma pasta comparti-

lhada no Google Drive[1], onde faremos referência a arquivos disponíveis em um determinado caminho.

Nas próximas seções deste capítulo, serão apresentados a justificativa para realização do trabalho, os objetivos gerais e específicos, a hipótese, contribuições e a estrutura da dissertação.

1.2 Justificativa

Neste trabalho foi utilizado o juiz *on-line* The Huxley[2] pelo fato de que foi disponibilizada uma API para conexão direta com a base de dados atual e a edição do código-fonte, permitindo acesso a diversas funções disponíveis no sistema, possibilitando assim a integração da solução e também extração de estatísticas. Esta solução permite que a aplicação seja mais dinâmica e rápida, já que outras alternativas exigiriam o acesso indireto via site web impossibilitando assim a interação e projeção de perspectivas futuras.

Ao realizar um estudo sobre os diversos juízes *on-line* e por fim escolher o The Huxley como ferramenta de estudo, foi realizada uma análise inicial em sua base de dados. Foi observado que as mensagens de *feedback* fornecidas pela ferramenta para erros de sintaxe em submissão tidas como erradas, são de difícil compreensão, muitas vezes não só para aprendizes de programação, mas também pelos professores. É importante ressaltar que os outros juízes *on-line* pesquisados também compartilham deste problema, já que apenas repassam para o usuário as mensagens emitidas pelos compiladores ou interpretadores. Essa situação ratifica a necessidade dos professores de utilizar juízes *on-line* que possuam a função de melhora nas mensagens de erros, eliminando assim um dos pontos de frustração e medo iniciais.

Diante do exposto, decidiu-se realizar um levantamento na base de dados, através da análise das submissões realizadas por todos os usuários do sistema, a fim de obter estatísticas que justifiquem a proposta deste trabalho.

Inicialmente foi criado um *script* de acesso à API da ferramenta, para acessar a base de submissões realizadas desde a sua concepção. A base encontra-se no caminho *Codes/baseErros.txt* e os *scripts* em *Codes/Cod_errorMsg.py* e *Codes/RequestData.py*, onde estes dois últimos fazem o acesso às mensagens de erros e às características das submissões, respectivamente. Estes dados serão explorados detalhadamente na Seção 3.1.

No momento da execução, em Janeiro de 2017, foram encontradas exatamente 667.836 submissões. A seguir, são ilustrados os resultados obtidos a partir das requisições realizadas à base do The Huxley.

A Figura 2 ilustra o percentual de acordo com os tipos de respostas das submissões, onde foi constatado que apenas 28,03% foram aceitas pelo sistema, obtendo resposta correta

[1] gg.gg/galileuFiles
[2] www.thehuxley.com

- "*CORRECT*", o que mostra que um montante de 71,97% são de submissões não aceitas ou incorretas, podendo ser possíveis fontes de erros e frustrações dos aprendizes ao tentarem resolver um determinado problema.

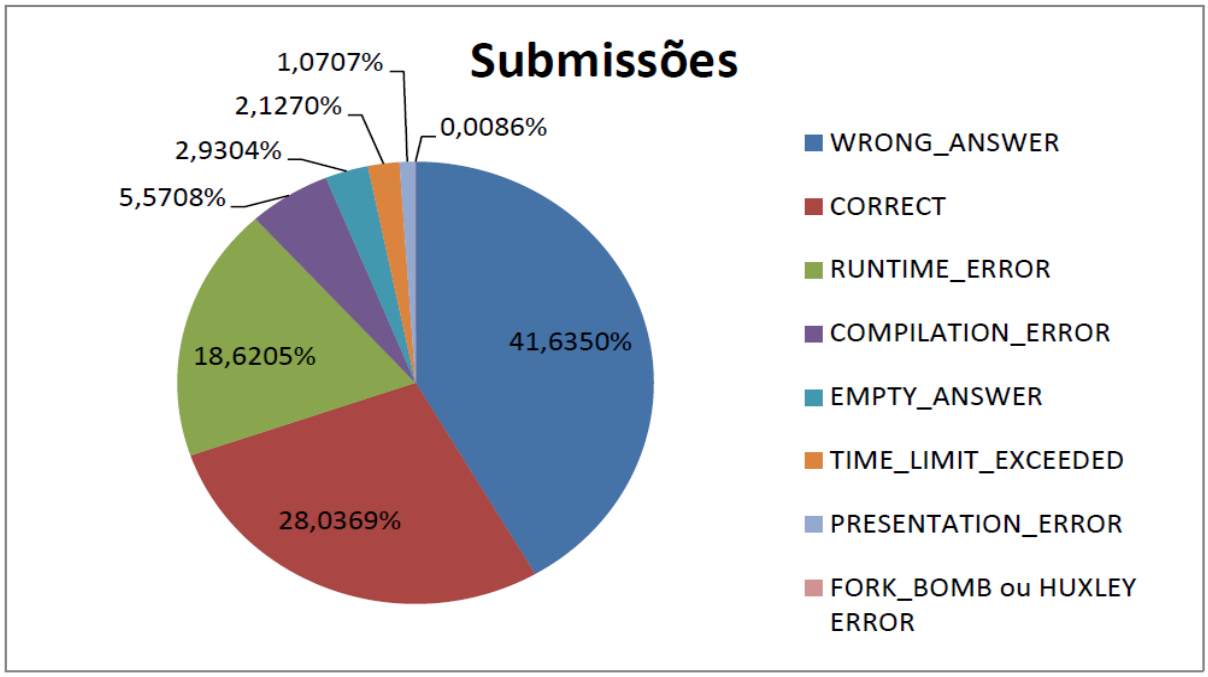

Figura 2 – Tipos de submissões do juiz *on-line* The Huxley.

Neste trabalho, foram estudados os erros de sintaxe, compilação e execução, correspondentes a 24,2% (*RUNTIME_ERROR (18,62%); COMPILATION_ERROR (5,57%) e FORK_BOMB ou HUXLEY_ERROR (0,008%)*) do total de submissões e a 33,62% das submissões erradas ao excluir as corretas, ou seja, de um total de 71,97% das submissões. Não sendo analisados os 44,82% referente às respostas que não foram aceitas (*WRONG_ANSWER (41,63%); TIME_LIMIT _EXCEEDED (2,12%); PRESENTATION_ERROR (1,07%)*, visto que a ferramenta já oferece a opção de correção baseando-se em casos de testes, sinalizando que a solução está incorreta ou o problema não foi entendido corretamente, ficando a cargo do usuário reestruturar sua solução para atender as especificações. Estes erros serão melhor contextualizados na seção 2.2.

A Figura 3, ilustra o percentual de submissões por linguagem, onde a linguagem C possui o maior percentual de submissões, com 53,44%, seguida pela linguagem Python, com 27,48%, sucedidas pelas linguagens: C++ com 7,88%, Java com 6,42%, Octave com 4,20% e por fim, Pascal com 0,58%.

Como ilustrado na Figura 2, 71,97% das submissões são consideradas como erradas. Deste total, a Figura 4 ilustra as submissões corretas e erradas de acordo com cada linguagem possível para submissão no The Huxley. A linguagem de programação Python se destaca, já que concentra 82,13% das submissões erradas contra 17,87% corretas, isso significa que de todas as submissões, o maior percentual não aceito por linguagem é desta. Em segundo lugar está

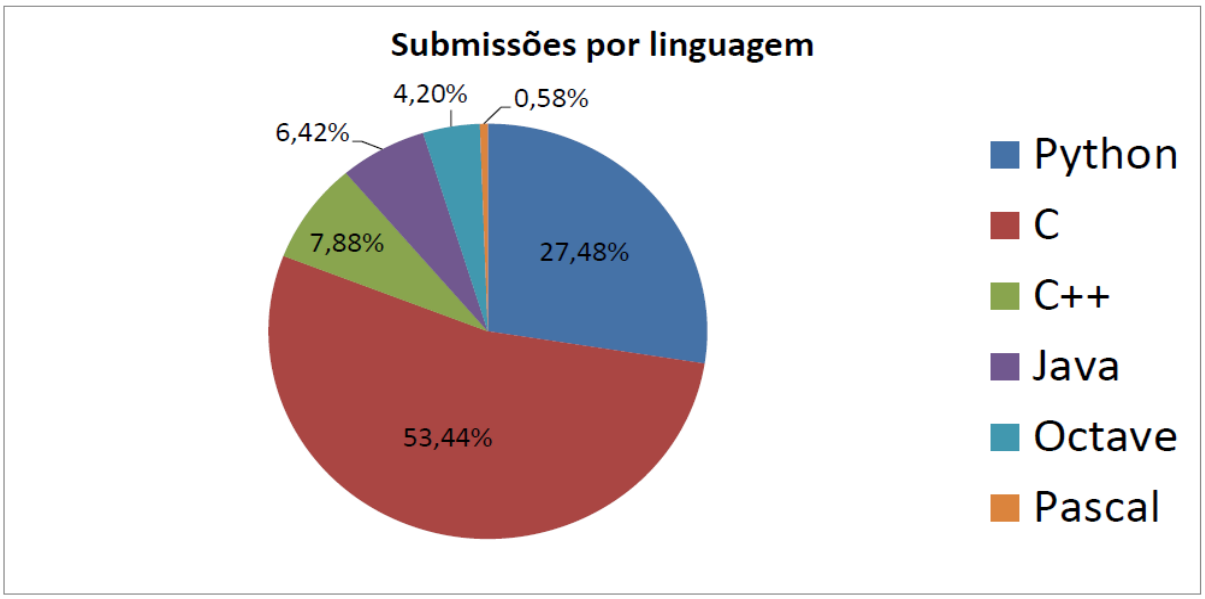

Figura 3 – Submissões por linguagem.

a linguagem Octave, com 72% consideradas erradas contra 28% corretas. A terceira posição é ocupada pela linguagem Java, com 68,48% erradas e 31,52% corretas. A quarta posição é da linguagem C, com 68,45% erradas e 31,55% corretas. Seguida pela linguagem Pascal com 65,83% erradas e 34,17% corretas e a linguagem C++ com 63,48% erradas e 36,52% corretas.

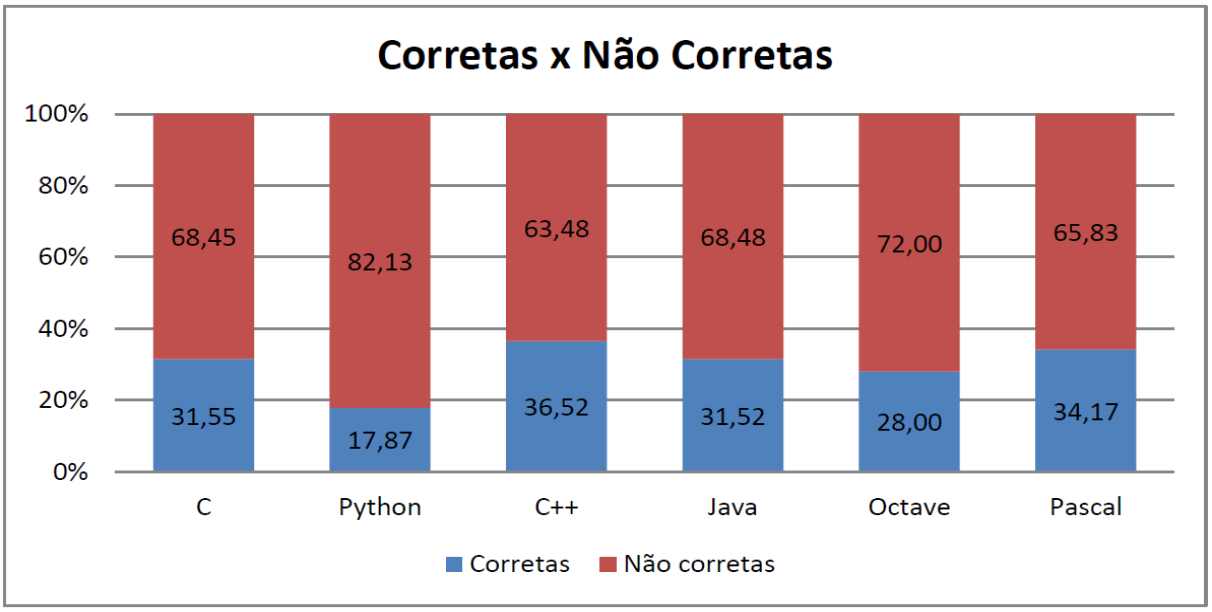

Figura 4 – Submissões consideradas erradas e não corretas de acordo com cada linguagem.

A Figura 5, ilustra o percentual de submissões corretas do total por linguagem, representando um total de 29,70%, onde temos que a linguagem C possui um maior percentual de submissões corretas, com 59,07%, seguida pela linguagem Python, com 17,87%, linguagens: C++ com 10,49%, Java com 7,63%, Octave com 4,12% e por fim, Pascal com 0,82%.

Como a linguagem Python possui maior percentual de submissões erradas, conforme

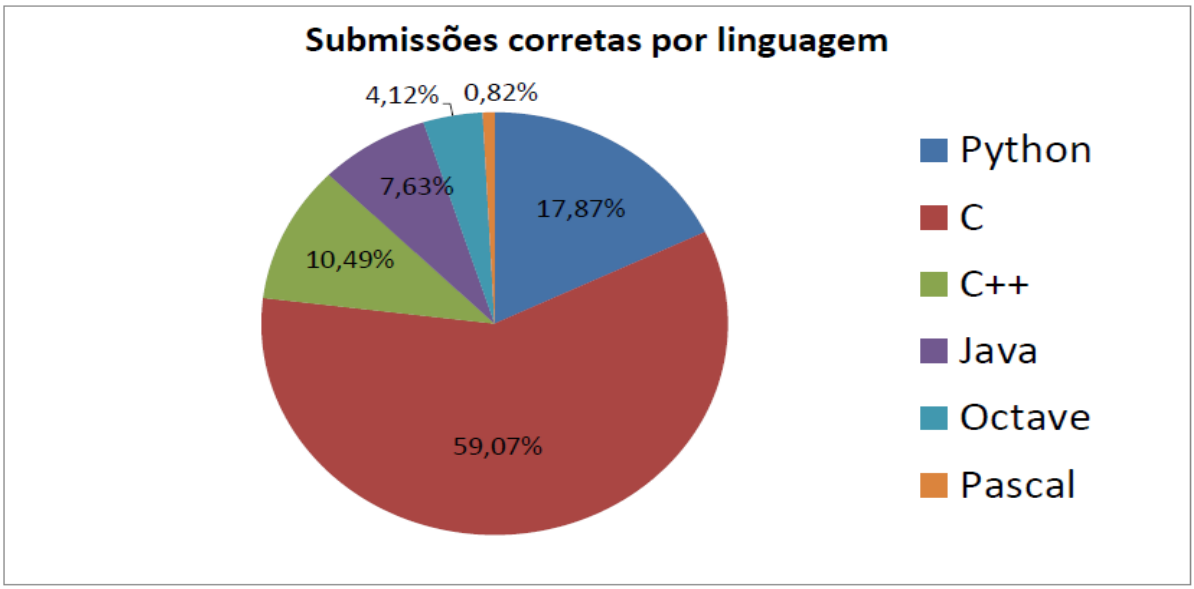

Figura 5 – Submissões consideradas corretas de acordo com cada linguagem.

ilustrado na Figura 4, mesmo não sendo a linguagem com maior quantidade de submissões, como ilustra a Figura 3, foi escolhida como objeto de estudo. Ao analisar os resultados das submissões desta linguagem, obtivemos que a maior porcentagem de submissões são de *RUNTIME_ERROR*, com cerca de 54,33%, caracterizando as mensagens de erros retornadas, seguidas pelos percentuais: *WRONG_ANSWER* com 23,90%; *CORRECT* com 17,98%; *EMPTY_ANSWER* com 2,03%; *PRESENTATION_ERROR* com 0,94%; *TIME_LIMIT_EXCEEDED* com 0,79% e por fim, *FORK_BOMB* com 0,0032%, como ilustra a Figura 6. O erro COMPILATION_ERROR não entrou na análise pois a linguagem Python não é compilada e não possui este tipo, pois os erros sintáticos são classificados como RUNTIME_ERROR.

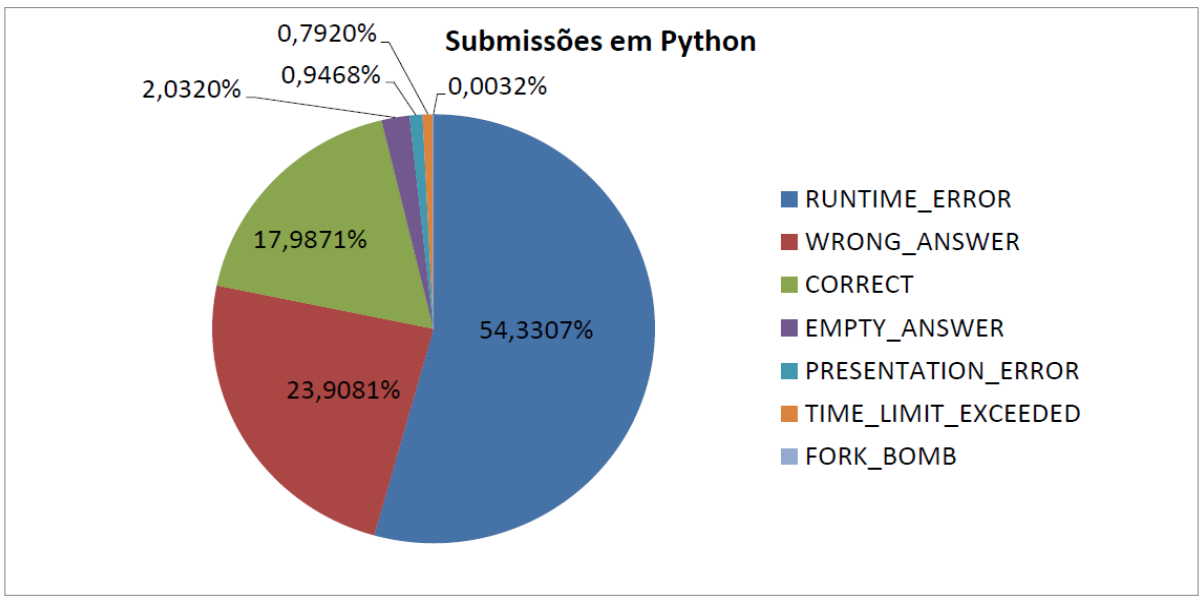

Figura 6 – Resultado das submissões realizadas em Python.

1.3 Objetivos

O objetivo geral deste trabalho consiste em propor uma abordagem que proporcione melhorias no processo de ensino e aprendizagem, provendo mensagens de erros de sintaxe que sejam facilmente compreendidas no contexto de aprendizes de programação de computadores em disciplinas introdutórias, contribuindo na identificação da causa do erro, para reduzir sua frustração inicial através de uma experiência construtivista de educação em um ambiente virtual de aprendizagem, que tem um juiz *on-line* como apoio à aprendizagem de programação.

1.3.1 Objetivos Específicos

Foram estabelecidos os seguintes objetivos específicos:

- Analisar a base de dados de submissões do juiz *on-line* The Huxley;
- Encontrar padrões de erros na base de dados de submissões do The Huxley;
- Especificar os requisitos iniciais da ferramenta a ser desenvolvida;
- Desenvolver um *Web Service* (Serviço Web) para fornecer mensagens de erro mais claras e úteis;
- Integrar a ferramenta desenvolvida ao The Huxley, customizando-o para dar suporte ao novo recurso de exibição de mensagens de *feedback* para as submissões erradas na linguagem Python;
- Avaliar a ferramenta através de um experimento controlado com participantes de turmas reais em um curso de graduação.

1.4 Hipótese

Considerando a necessidade da utilização de ambientes virtuais de aprendizagem que possuam *feedback* de erros sintáticos mais entendíveis por seus usuários, com a finalidade de avaliar a utilização desta abordagem como apoio ao ensino e aprendizado de programação, tem-se como hipótese: A utilização de mensagens de erros sintáticos mais entendíveis, permitirá uma maior eficiência na resolução deste erros, facilitando o ensino-aprendizagem de disciplinas iniciais de programação devido ao processo de correção ser mais rápido e fácil.

1.5 Contribuições

Com o desenvolvimento desta dissertação, destacam-se como principais as seguintes contribuições:

- Revisão dos principais trabalhos que abordam a melhoria de mensagens de erros para aprendizes de programação;
- Comparação dos trabalhos relacionados, observando critérios como: mensagens melhoradas e sua abrangência; linguagens disponíveis para realização de submissões; disponibilidade como AVA e WEB e por fim, se foi realizada alguma experimentação utilizando a ferramenta;
- Criação de serviço web que forneça mensagens melhoradas de acordo com o tipo do erro, sendo disponibilizado acesso para qualquer desenvolvedor;
- Integração com o juiz *on-line* The Huxley;
- Experimentação e aplicação em ambiente real de sala de aula com turmas de alunos de disciplinas introdutórias de programação de cursos de graduação.

1.6 Organização da Dissertação

Este trabalho está organizado em seis capítulos. Os próximos cinco capítulos estão divididos da seguinte forma:

- O Capítulo 2 apresenta a fundamentação teórica que embasa a realização deste trabalho, assim como os trabalhos relacionados e toda a instrumentação utilizada para sua realização;
- No Capítulo 3 é abordada a especificação deste trabalho, desde a elucidação de requisitos através da análise da base de dados até sua implementação e integração;
- No Capítulo 4 é apresentada a realização do estudo experimental através de sua aplicação em turmas iniciais de programação;
- No Capítulo 5 é apresentada a análise dos resultados obtidos através da experimentação descrita no capítulo anterior, mostrando as análises estatísticas, que consiste tanto de testes estatísticos para validação de hipóteses, como de estatística descritiva dos dados;
- Por fim, o Capítulo 6 traz as conclusões e os trabalhos futuros.

2
Fundamentação Teórica

Neste capítulo são apresentados os conceitos teóricos necessários para fundamentar a realização deste trabalho. Realizou-se um levantamento acerca dos principais trabalhos que abordam uma temática relacionada. Na primeira seção são apresentados os conceitos de Ambiente Virtual de Aprendizagem. A segunda seção aborda juízes *on-line*. Na terceira são ilustradas as ferramentas utilizadas para efetivação deste trabalho. Por fim, na quarta e última seção, são apresentados os trabalhos relacionados.

2.1 Ambiente Virtual de Aprendizagem

Segundo Cardoso (2010), um Ambiente Virtual de Aprendizagem (AVA) consiste num sistema de computador em rede ou na Internet que possui um conjunto de ferramentas e recursos tecnológicos integrados, os quais permitem mediar o processo ensino-aprendizagem seja ele presencial, a distância ou semipresencial. Esta ferramenta de ensino vem crescendo não só no que tange à educação a distância, mas também na educação presencial e semipresencial. Alguns deles vem causando grande impacto no público de usuários e em diferentes instituições, assim como nos diversos modelos de ensino presentes.

Os principais recursos tecnológicos geralmente utilizados nesses ambientes podem ser agrupados em quatro eixos, de acordo com Pereira, Schmitt e Dias (2007) e ilustrado na Figura 7:

- Informação e documentação (permite apresentar as informações institucionais do curso, veicular conteúdos e materiais didáticos, fazer *upload* e *download* de arquivos e oferecer suporte ao uso do ambiente);
- Comunicação (facilita a comunicação síncrona e assíncrona);

- Gerenciamento pedagógico e administrativo (permite acessar as avaliações e o desempenho dos aprendizes, consultar a secretaria virtual do curso, entre outros);
- Produção (permite o desenvolvimento de atividades e resoluções de problemas dentro do ambiente).

Figura 7 – Principais eixos de Ambientes Virtuais de Aprendizagem. Figura extraída de (PEREIRA; SCHMITT; DIAS, 2007).

Existem diversos AVA's, mas, segundo Ribeiro, Mendonça e Mendonça (2007) e PENTERICH (2005) os mais comuns são TelEduc[1], AulaNet[2], Amadeus[3], Dokeos[4], e-ProInfo[5], Edmodo[6]. O Moodle ou *Modular Object Oriented Distance Learning*, é uma plataforma de aprendizagem projetada para fornecer aos educadores, administradores e alunos um sistema integrado para criar ambientes de aprendizagem personalizados. Ele é um software livre que pode ser usado para criar sites web interativos onde professores e estudantes podem se comunicar e colaborar em modelos educacionais. Atualmente ele está sendo utilizado em 224 países sendo que o Brasil é o terceiro país com mais registro de utilização. Várias instituições de ensino no Brasil utilizam o Moodle como AVA para suporte educacional (CARDOSO, 2010) (MOODLE, 2016).

[1] www.teleduc.org.br
[2] web.ccead.puc-rio.br/aulanet2
[3] softwarepublico.gov.br/social/amadeus
[4] www.dokeos.com
[5] eproinfo.mec.gov.br
[6] www.edmodo.com

Em termos de AVA, o The Huxley também atua no eixo de comunicação, já que permite a troca de mensagens entre alunos e professores, assim como de informação e documentação referentes a problemas de programação, facilitando o gerenciamento pedagógico e administrativo de turmas ou grupos, visto que possui mecanismo de consulta de desempenho de alunos, assim como atribuição de notas a exercícios. Possui também o eixo de produção, permitindo o desenvolvimento de atividades e resolução de problemas dentro do ambiente.

2.2 Juízes *on-line*

Normalmente a avaliação automática em atividades é feita estritamente em questões de múltiplas escolhas, visto que questões discursivas possuem um grande número de respostas possíveis, sendo computacionalmente impossível prevê-las em sua totalidade.

Avaliação automática já vem sendo discutida em grandes debates de educação, sendo um problema antigo de acordo com Kay et al. (1994). Segundo Lino et al. (2007), os professores enfrentam cotidianamente a dificuldade em avaliar manualmente exercícios de programação, principalmente em turmas grandes, dificultando muito a correção em curto prazo de forma que possibilite ao estudante aperfeiçoar sua solução.

Em exercícios de programação é comum haver mais de uma solução para um determinado problema. Portanto, torna-se difícil para o educador avaliar individualmente todas as soluções de um determinado exercício e pontuar de acordo com a quantidade de acertos ou erros dos estudantes. Geralmente, exercícios de programação são avaliados levando-se em consideração duas métricas: 1) se o algoritmo retorna o resultado esperado; e 2) a complexidade da solução, isto é, se a solução está bem escrita(MOREIRA; FAVERO, 2009).

Juízes *on-line* são sistemas que fornecem um repositório de diversos problemas de programação e estão disponíveis para que os usuários possam submeter as possíveis soluções destes em forma de código-fonte, em uma determinada linguagem de programação. Estas soluções são submetidas e passadas por uma bateria de casos de testes, onde são executados exemplos de entradas - não visíveis aos usuários, gerando então saídas, as quais são comparadas com a solução correta. Desta forma, é retornada a respectiva resposta julgada de acordo com determinada execução, podendo possuir as seguintes designações de acordo com a Tabela 1: (KURNIA; LIM; CHEANG, 2001) (PRIOR, 2003) (SANTOS; RIBEIRO, 2011) (SILVA, 2016).

Tabela 1 – Tipos de respostas dos juízes *on-line*. Tabela adaptada de Silva (2016)

Resposta	**Descrição**
YES	Também conhecido como *ACCEPT*. Indica que seu programa foi aceito.
NO: Incorrect Output	Também conhecido como *Wrong Answer*. Indica que seu programa respondeu incorretamente a pelo menos um dos casos de teste dos juízes.
NO: Time Limit Exceeded	A execução do seu programa excedeu o tempo permitido pelos juízes para um determinado problema.
NO: Runtime Error	Durante o teste ocorreu um erro de execução (causado pelo seu programa) na máquina dos juízes. Acesso a posições irregulares de memória ou estouro dos limites da máquina são os erros mais comuns.
NO: Compilation Error	Seu programa tem erros de sintaxe. Pode ser ainda que você errou o nome do problema ou a linguagem no momento da submissão.
NO: Output Format Error	Também conhecido como *Presentation Error*, indica que a saída do seu programa não segue a especificação exigida, apesar do "resultado" estar correto.

Algumas ferramentas foram desenvolvidas usando estes conceitos. Segundo Tonin e Bez (2013), por exemplo, temos o URI Online Judge, que é uma ferramenta de correção automática de exercícios de programação, assim como The Huxley, UVA Online Judge, SPOJ Online Judge e outros (PAES et al., 2013) (BEZ; TONIN; RODEGHERI, 2014a).

De acordo com Kjöllerström (2012) e Moreira e Favero (2009), os principais benefícios ao se adotar um sistema de correção *on-line*, são:

- Menor esforço, uma vez que conta com o auxílio da ferramenta;
- Melhor administração dos estudantes e de suas tarefas;
- Melhor rastreamento individual dos estudantes;
- Melhor qualidade de ensino, devido ao maior tempo de prática;
- Mais tempo para contato com os estudantes.

Ao utilizar um juiz *on-line*, os alunos obtêm o *feedback* automaticamente de uma submissão a um determinado problema. Quando o juiz *on-line* está acoplado a um AVA, o professor pode acompanhar o rendimento individual de cada um dos alunos, possibilitando práticas efetivas de programação com *feedback* em tempo real. (BEZ; TONIN; RODEGHERI, 2014b; PAES et al., 2013).

2.2.1 The Huxley

O The Huxley, segundo Paes et al. (2013), é uma ferramenta web que permite submeter código-fonte em diversas linguagens de programação, para atividades práticas de aprendizagem de programação, baseando-se em um repositório de centenas de problemas, destacando-se por possuir funcionalidades acadêmicas, como controle de turmas, grupos, atividades e gerenciamento de desempenho dos alunos, disponibilizados para os professores. Para cada submissão, o aluno recebe *feedback* da correção automática pelo sistema através de análise sintática do código e testes de aceitação.

Algumas funcionalidades são:

- Um usuário/aluno realiza submissões para os problemas;
- Um usuário/aluno faz parte de um grupo;
- A um grupo estão associados questionários, que por sua vez agregam problemas e possuem uma nota específica;
- O questionário está associado a um determinado grupo;
- Os resultados de submissões são visíveis por todos os usuários.

A Figura 8 ilustra o funcionamento geral da ferramenta, onde o servidor está na nuvem, armazenando as centenas de problemas e submissões dos alunos, assim como todas as suas funcionalidades.

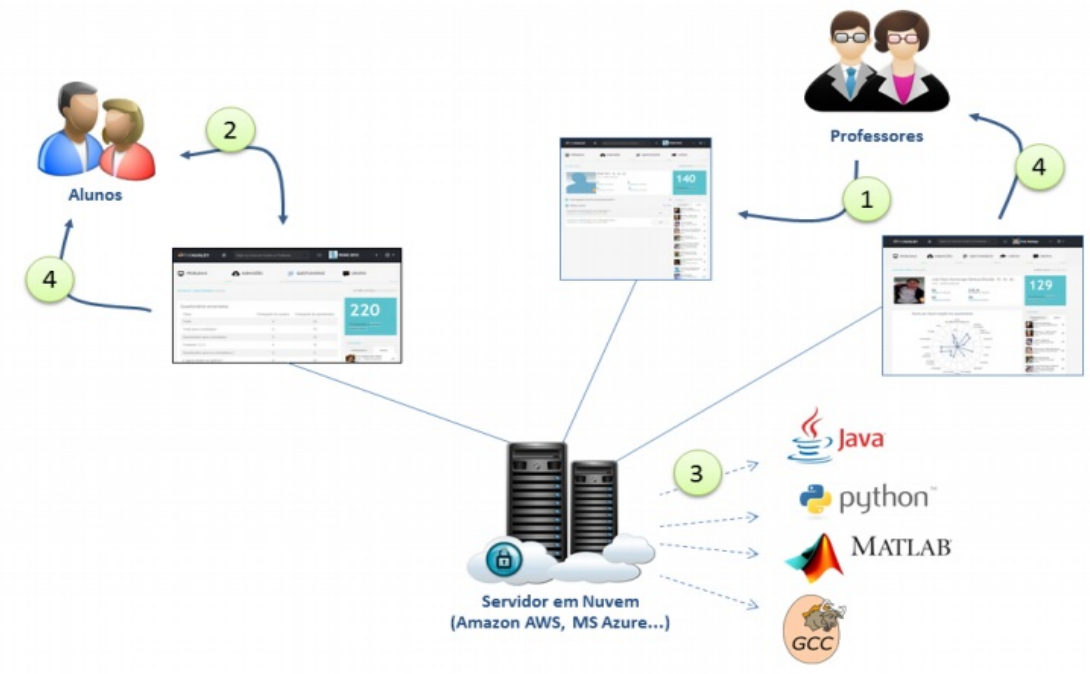

Figura 8 – Visão geral de funcionamento do The Huxley, imagem extraída de (PAES et al., 2013).

Segundo Paes et al. (2013), o professor acompanha o desempenho de seus alunos de acordo com: quantidade de problemas resolvidos, porcentagem de acertos/erros, tipos de problemas com mais erros, detecção de plágio e erros específicos de cada aluno.

Ainda de acordo com Paes et al. (2013), além dessas características, possui também:

- Acompanhamento das atividades do aluno pelo professor;
- Problemas separados por categorias e por níveis de dificuldades;
- *Ranking* por problema e por linguagem de programação;
- Visualização e edição do código-fonte diretamente no navegador;
- Visualização das linhas do código-fonte com erro quando recebe como resposta o Erro de Compilação;
- Visualização de detalhes sobre erros encontrados durante a execução da solução;
- Indicação do percentual de casos de teste que falharam, quando da submissão de uma solução incorreta para julgamento;
- Avaliações automatizadas, ou seja, é possível definir data e hora de início de disponibilização da prova, bem como seu tempo de expiração;
- Histórico de submissão de uma solução realizada pelo aluno.

Como mostrado na Seção 2.2, os juízes *on-line* possuem mensagens específicas de resposta às submissões. O The Huxley segue este padrão, porém acrescido de algumas outras mensagens, como EMPTY_ANSWER, sinalizando que a resposta retornada pela execução do código-fonte foi vazia e FORK_BOMB ou HUXLEY_ERROR, sinalizando que ocorreu erro no servidor do sistema The Huxley ao tentar julgar uma submissão.

2.3 Instrumentação

Nesta seção são apresentadas as ferramentas e recursos utilizadas para concretização deste trabalho.

2.3.1 Python e IDEs

Python é uma linguagem de programação interpretada, interativa e orientada a objetos. Fornece estruturas de dados de alto nível, tais como listas, matrizes, dicionários de dados, vinculação dinâmica, módulos, classes, exceções, gerenciamento de memória automática e outros. Possui uma sintaxe simples. Foi projetada em 1990 por Guido Van Rossum. Assim como

outras linguagens, tem licença gratuita, mesmo para fins comerciais. Um programa Python é compilado automaticamente e interpretado pela plataforma, transformando-o em *byte code*, que então é interpretado (SANNER et al., 1999).

Neste trabalho foi utilizada esta linguagem devido e sua sintaxe agradável e simples. Para execução dos códigos criados e testes, foram utilizadas as ferramentas *IDLE Python* e Eclipse, possibilitando o uso de recursos e bibliotecas da mesma. Para obtenção da base de dados de submissões, foi feito o uso de recursos disponíveis na linguagem, através de requisições *Web*, bem como toda a análise da base, o que será melhor discutido na Seção 3. Além disso, o serviço *Web* desenvolvido para dar acesso à abordagem proposta foi codificado na linguagem Python.

2.3.2 Bitbucket

O Bitbucket[7] é um serviço de hospedagem baseado na *Web*, sendo de propriedade da Atlassian. É usado para código-fonte e projetos de desenvolvimento que utilizam os sistemas de controle de versão e revisão. Oferece planos comerciais e contas gratuitas, sendo um número ilimitado de repositórios privados, porém limitando a cinco usuários por cada um. A existência destes planos favoreceu a escolha deste serviço, visto que fornece o controle de versão grátis para este trabalho, além do recurso de repositório privado, garantindo privacidade, segurança e agilidade no desenvolvimento. É semelhante ao GitHub[8], porém não oferece gratuitamente repositórios privados.

2.3.3 Postman

O Postman[9] é uma ferramenta completa para desenvolvimento de *API - Application Programming Interface*, foi projetado para apoiar diversos aspectos do desenvolvimento de aplicações, simulando o funcionamento real das mesmas. Possui uma única camada subjacente, o que garante um melhor desempenho e maior experiência de usuário, assim como recursos tais como: requisições, testes e *scripts* de pré-requisição, variáveis, ambientes e descrições de requisições, projetados para funcionar perfeitamente juntos. Em síntese, simula os diversos métodos do protocolo HTTP, tais como: POST, DELETE, GET, PUT e outros.

Foi utilizado neste trabalho com a finalidade de testar a ferramenta desenvolvida, simulando as requisições dos usuários em um serviço *Web* local, sendo possível verificar como a mesma se comportava ao ser submetida às requisições.

[7] bitbucket.org/
[8] github.com
[9] www.getpostman.com

2.3.4 Excel

O Excel[10] é um software desenvolvido pela empresa Microsoft, usado por diversas empresas privadas ou não, para realização de operações financeiras e contábeis. O aplicativo Excel é usado para realizar uma infinidade de tarefas como: cálculos simples e complexos, criação de lista de dados, elaboração de relatórios e gráficos sofisticados, projeções e análise de tendências, análises estatísticas e financeiras, além de trazer incorporado uma linguagem de programação baseada em Visual Basic. É uma ferramenta muito utilizada para realização de análises de dados, agrupamento e organização. Neste trabalho foi feito seu uso com esta finalidade, o que será melhor discutido no Capítulo 3.

2.3.5 R Studio

R Studio é um software com licença gratuita para desenvolvimento de programas para análises estatísticas e gráficas na linguagem R. É muito utilizado para gerenciamento e manipulação de dados, simulação e modelagem estatística. R é uma linguagem de alto nível com o objetivo de desenvolver métodos computacionais de forma simples e efetiva, possibilitando um desenvolvimento ativo e visualização gráfica de ponta dos dados. Além disso, seu código fonte é livre (STUDIO, 2017b).

Neste trabalho foi utilizada esta linguagem como ferramenta para realização de testes estatísticos sobre os dados finais do experimento, assim como geração de gráficos dos mesmos. Uma discussão mais aprofundada encontra-se no Capítulo 5.

2.3.6 Ferramentas da Google

A Google disponibiliza ferramentas *on-line*, sem que seja preciso instalar programas e geralmente são gratuitos. Além disso, permite o acesso aos documentos a partir de qualquer computador, sendo possível compartilhar e editar com e por várias pessoas, em tempo real (MACHADO, 2009).

Neste trabalho foram utilizados os serviços: Documentos, Planilhas, Sites e Formulários, o que permitiu a interação e o intercâmbio de ideias, através da possibilidade de trocar informações e interferir nos processos de desenvolvimento e organização do trabalho, o que será melhor apresentado na Seção 3.

2.4 Trabalhos relacionados

Nesta seção são apresentados trabalhos relacionados à problemática discutida nesta proposta. Algumas pesquisas mostram que há muito interesse no que diz respeito à aplicação de

[10] products.office.com/pt-br/excel

exibição de dicas e mensagens melhoradas para apoiar o ensino de programação de computadores, conforme ilustram os trabalhos a seguir.

2.4.1 CAP

O Code Analyzer for Pascal (CAP) (SCHORSCH, 1995), foi desenvolvido na Academia da Força Aérea do Estados Unidos para proporcionar aos alunos um *feedback* amigável e automatizado sobre erros comuns de sintaxe, lógica e de estilo dos programas Pascal. Tais erros foram recolhidos e classificados através de um estudo informal de todos os 520 alunos matriculados no curso de Introdução à Ciência da Computação - CS110.

CAP faz uma análise estática do código e informa ao estudante o que está errado, porque está errado e como corrigir o problema específico, podendo relatar 111 mensagens de diagnóstico diferentes (incluindo 62 para erros de sintaxe, 21 para erros de lógica e 28 para erros de estilo).

Os instrutores notaram que com o uso do CAP o número de erros de sintaxe e lógica triviais que os alunos pediam ajuda diminuíram, poupando-lhes muito tempo. De forma análoga, a qualidade dos programas dos estudantes aumentou.

CAP foi útil para os alunos, identificando e ajudando-os a corrigir alguns tipos de erro que anteriormente eram fontes de frustração, mas alguns alunos ignoraram o hábito de programar seguindo as regras de estilos da linguagem Pascal, pois sabiam que o CAP iria indicar as correções necessárias. Como consequência, estes alunos tornaram-se dependentes da ferramenta e não conseguiam programar sem o apoio do CAP.

As Figuras 9, 10 e 11 esboçam exemplos de mensagens geradas pela ferramenta ao serem exibidos, respectivamente, erros de sintaxe, lógico e de estilo.

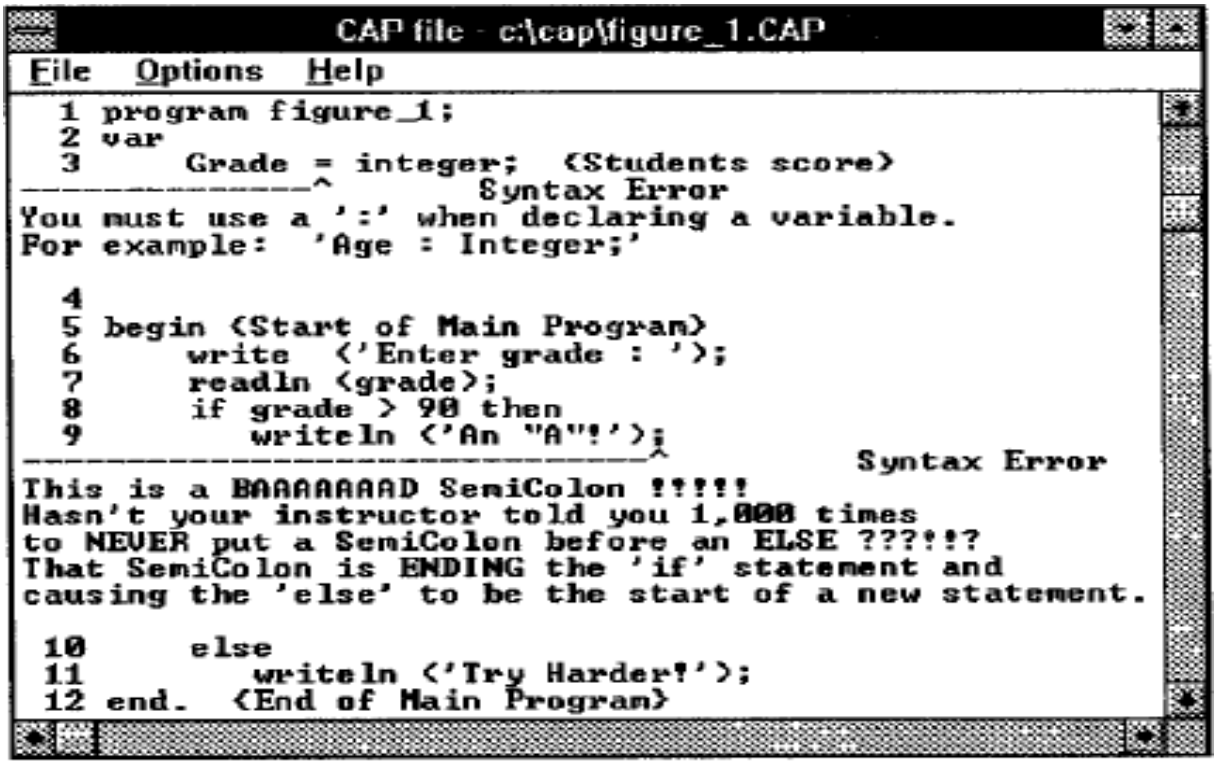

Figura 9 – Mensagem de erro de sintaxe ao utilizar a ferramenta CAP. Figura extraída de (SCHORSCH, 1995).

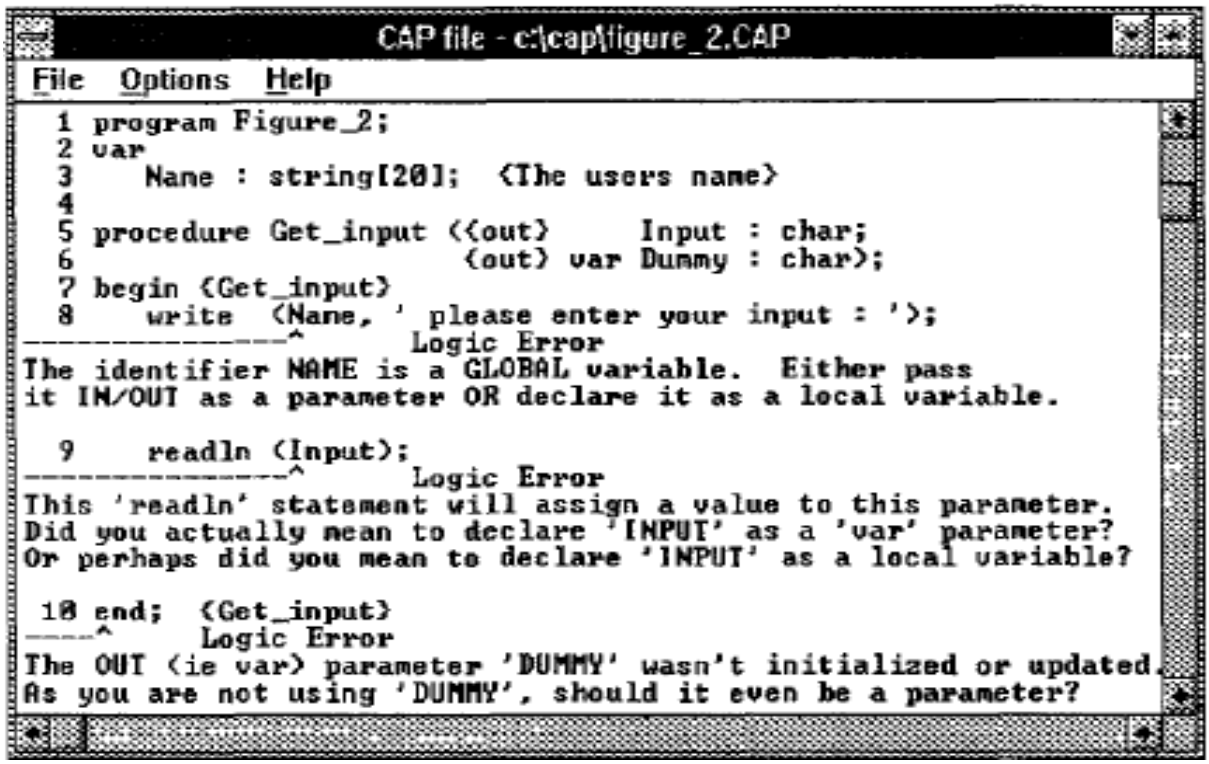

Figura 10 – Mensagem de erro lógico ao utilizar a ferramenta CAP. Figura extraída de (SCHORSCH, 1995).

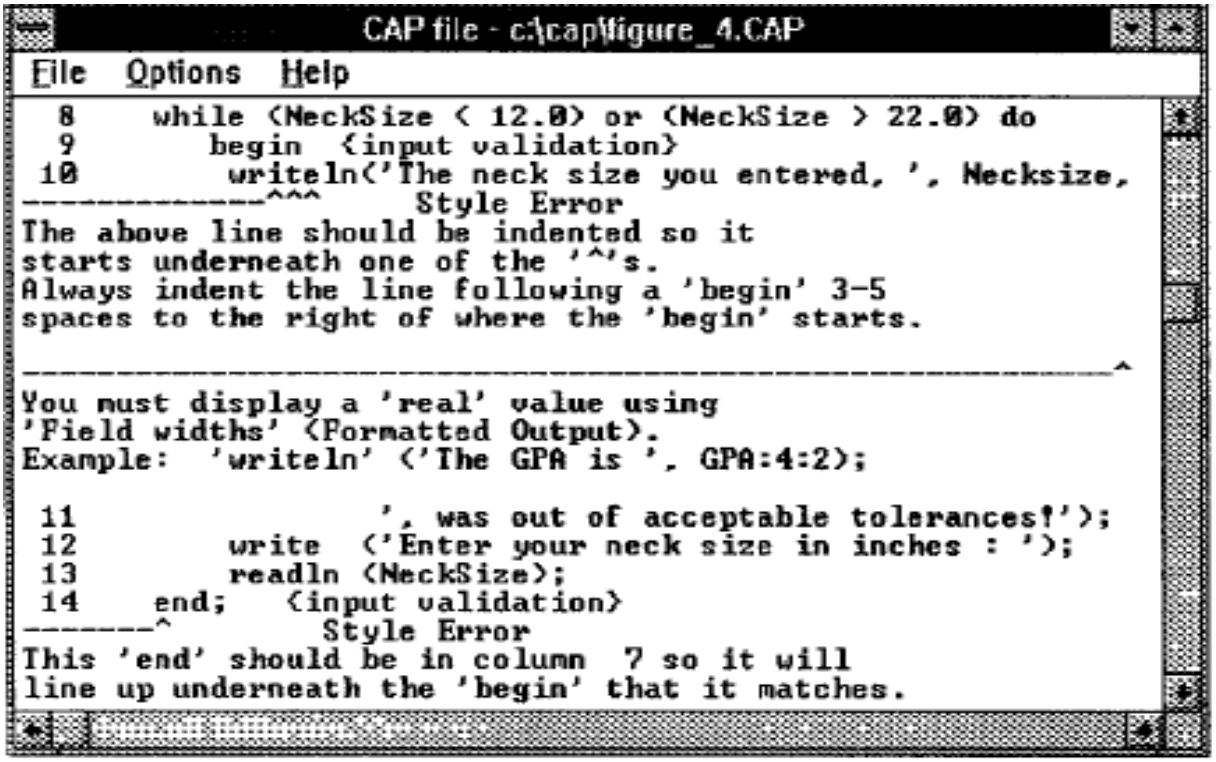

Figura 11 – Erro de estilo ao utilizar a ferramenta CAP. Figura extraída de (SCHORSCH, 1995).

As limitações do CAP são: é muito antigo, pois foi feito no ano de 1995, apesar de não ser uma limitação, porém não foi continuado, apoiando apenas a linguagem Pascal e, finalmente, não estar disponível como um sistema *on-line*.

2.4.2 InSTEP

O Independent Student Tutoring by Examining Programs (InSTEP) (ODEKIRK-HASH; ZACHARY, 2001), é um sistema de tutor *on-line* para programadores iniciantes da linguagem C, que tem como motivação orientar os alunos através de uma aprendizagem construtiva. Nele é disponibilizada uma lista de exercícios de programação, onde cada problema tem seu código apresentado em um formulário web com partes fixas e outras partes com campos que os alunos devem completar.

InSTEP limita as partes modificáveis do código, para não ter uma grande variabilidade de programa, permitindo assim que ele localize e analise mais facilmente a maioria dos problemas de um campo, apontando os possíveis erros e sugerindo suas correções.

Um estudo educacional foi realizado para avaliar o INSTEP, em uma turma de Engenharia de 120 alunos da Universidade de Utah, nos quais 66 deles participaram. Os alunos foram separados em 3 grupos: os que tiveram o auxilio total da ferramenta; os que a usavam parcialmente só informando se o problema estava certo ou errado sem a análise e sugestões; e os que não a usaram.

Foram analisadas as várias medições, tentando correlacionar as diferenças entre os três grupos, e não se obteve resultados significativos, visto que os códigos que eram para ser completados foram bastante simples e não requereram programação complexa. Entretanto, concluiu-se que o programa foi útil para poupar tempo e auxiliar os tutores humanos em um *feedback* mais rápido, permitindo que os tutores se concentrassem nos estudantes que precisou de mais ajuda.

A listagem 2.1 ilustra um exemplo de código-fonte na linguagem C ao ser executado na ferramenta, produz o *feedback* ilustrado na listagem 2.2, onde as mensagens de erros são melhoradas. Embora o código tenha compilado corretamente, as mensagens são de acordo com o que foi gerado na saída da execução do código, mostrando uma explicação mais detalhada de erros com base nas saídas obtidas e esperadas. Portanto, não aborda o problema de erros sintáticos.

```c
/*
  Counting from 1 to 10 in a While Loop
*/
#include<stdio.h>

int main() {
  int counter;
  printf("Answer Section:\n");
  counter = 1;
  while (counter < 10) {
    printf( "%d\n", counter);
    counter = counter - 1;
  }
  return(0);
}
```

Listagem 2.1 – Código exemplo na linguagem C, executado na ferramenta InStep.

```
COMPILE
Program compiled correctly.

CODE CRITIQUE
Your program does not output the correct output.

Your process created more then 50 lines of output, which means you have an
    infinite loop.

Check the place where you change the loop variable within the loop.

PROGRAM OUTPUT
Answer section:
1
0
```

```
15 -1
16 -2
17 -3
18 And more lines of output
```

Listagem 2.2 – Exemplo de saída gerada pela ferramenta InStep ao executar o código da Listagem 2.1.

2.4.3 Expresso

Os erros dados pelo compilador são muitos enigmáticos e não ajudam muito no processo de aprendizagem dos estudantes. Foi pensando nisso que foi desenvolvida a ferramenta Expresso (HRISTOVA et al., 2003), com o propósito de gerar melhores mensagens de erros do que as existentes nos compiladores e também fornecer sugestões sobre como corrigir o código-fonte.

Foi elaborada uma lista com 62 erros de programação da linguagem Java, dos quais 20 foram identificados como essenciais. Esta seleção foi feita baseando-se nos relatos de tutores que ensinam Java em universidades e de professores de ciência da computação de 58 escolas listadas no *US News and World Report's Top 50 Liberal Arts Colleges for 2002* e membros da *Special Interest Group on Computer Science Education* (SIGCSE) da *Association for Computing Machinery*.

Os erros foram classificados em três categorias: erros de sintaxe (referem-se a erros de ortografia, pontuação e ordem das palavras no programa); erros de semântica (lida com o significado do código) e erros lógicos (erros que surgem a partir do pensamento equivocado do programador).

Foi implementado um pré-processador de três passos: o primeiro passo remove comentários, mantendo o controle dos números da linhas e salvando os caracteres resultantes em um vetor; O segundo passo remove espaços em branco, salvando o resultando em um vetor de strings; O terceiro passo detecta os erros sintáticos, imprimindo uma mensagem de erro apropriada.

Como limitação deste trabalho pode-se citar a obsolescência, já que foi desenvolvida no ano de 2003 e não houve continuidade. Além disso, como não houve avaliação da ferramenta em um ambiente real de sala de aula, não foi confirmado se as habilidades de programação dos alunos mudaram ao se utilizar esta ferramenta. Os autores listam esta avaliação da ferramenta como trabalho futuro, porém nenhuma publicação adicional foi encontrada.

A Listagem 2.3 ilustra um exemplo de código-fonte na linguagem Java que ao ser executado na ferramenta, obtém o *feedback* ilustrado na Figura 12, onde as mensagens de erros são melhoradas.

```
1 import java.awt.*;
2 import java.applet.*;
3 import java.awt.event.*;
```

```
4 public class SampleRun extends Applet implements
5 ActionListener {
6   Label sampleLabel = new Label("Sample Label",
7   Label.LEFT);
8   Panel samplePanel = new Panel();
9
10  public void init( );{
11    samplePanel.setLayout(new
12    FlowLayout(FlowLayout.LEFT));
13    samplePanel.add(sampleLabel);
14    add(samplePanel);
15    repaint();
16  }
17
18  public void paint(Graphics g){
19    char bear = 'p';
20    String cheese = "appleSauce";
21    computeSum(cheese, bear);
22  }
23
24  public int computeSum(int apple, int sauce){
25    int appleSauce;
26    if (apple = sauce){
27    appleSauce= apple + sauce;
28    }
29    else if (apple > sauce);{
30    appleSauce == apple - sauce;
31    }
32    else{
33    appleSauce = 49;
34    }
35    return appleSauce;
36  }
37 }
```

Listagem 2.3 – Código exemplo executado na ferramenta Expresso.

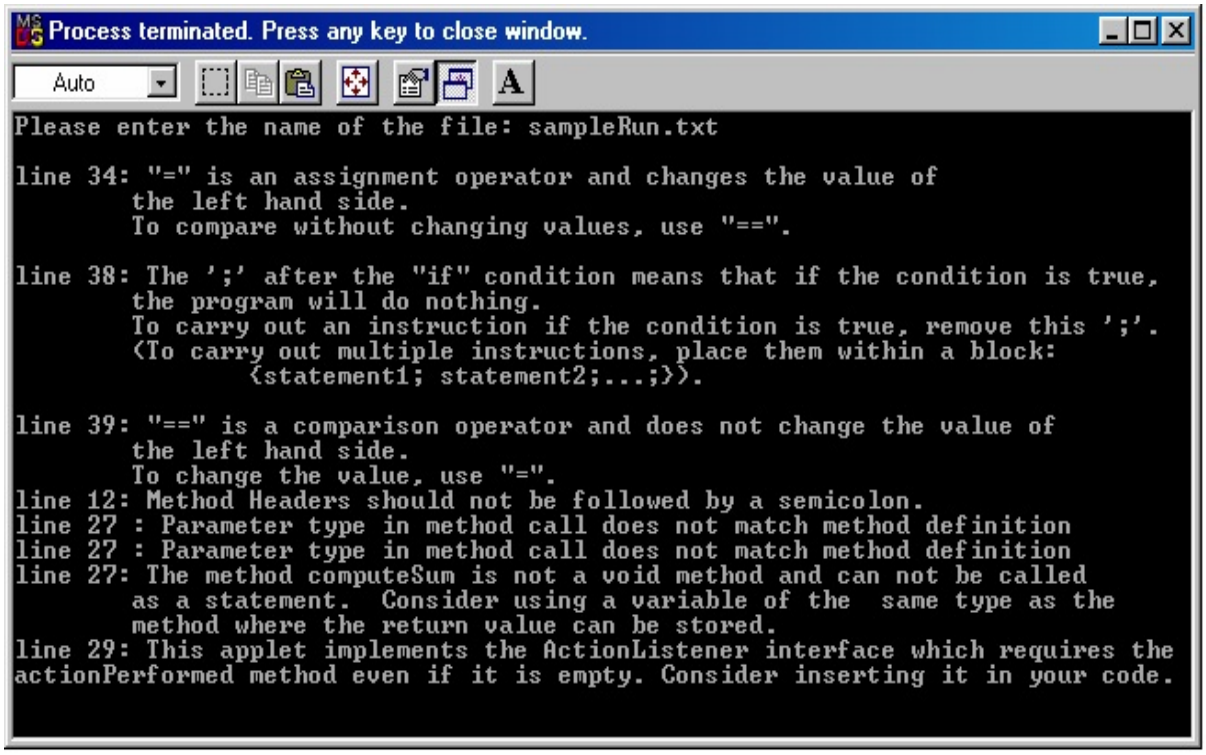

Figura 12 – Saída gerada ao executar o código da Listagem 2.3 na ferramenta Expresso. Figura extraída de (HRISTOVA et al., 2003).

2.4.4 Análise de Erros e Tempo de *Debug*

O trabalho de Ahmadzadeh, Elliman e Higgins (2005) tenta refinar os métodos de ensino atuais, aprendendo com os erros dos alunos. Foi utilizada a linguagem Java e análise dos dados foram separadas em duas fases: na primeira, as mensagens do compilador foram coletadas utilizando um compilador chamado Jikes (*open source*), junto a um editor chamado JCreator. Na segunda o compilador foi modificado para produzir três tipos de erros: "*syntax errors*", "*semantic errors*"e "*lexical errors*", foram analisados 108.652 registros de erros dos estudantes que teve como resultado 36 *syntax errors*, 63 *semantic errors* e 1 *lexical errors*. O erro mais comum encontrado foi o de não definir uma variável.

Foi realizada uma análise da correlação linear entre o número de erros por compilação, o tempo necessário para depurar o programa, e a nota que foi obtida no exame, com o objetivo de verificar sua relação, sendo obtido o resultado de que não há correlação entre o número de erros e o tempo gasto para depurar, mas houve relação com o tempo gasto com a nota obtida. Foi constatado que aqueles que são mais rápidos para depurar, são mais eficientes em encontrar erros e capazes de alcançar notas mais altas.

Como limitação, destaca-se que não foi criada nenhuma ferramenta que use os conceitos dos tipos de erros analisados, isto é, foi realizado apenas um estudo para buscar evidências de que há uma correlação entre o número de erros e o tempo de depuração, porém não foram encontradas.

2.4.5 Avaliação da Efetividade das Mensagens de Erros

O trabalho de Marceau, Fisler e Krishnamurthi (2011) buscou analisar a efetividade das mensagens de erros exibidas pela ferramenta DrScheme. Tenta resolver o problema de que linguagens de programação são projetadas para especialistas, e não para o ensino de programação. As mensagens foram desenvolvidas ao longo de uma década através de observações em laboratórios e sala de aula. Mesmo assim, ainda é comum perceber problemas de interpretações de mensagens.

Esta ferramenta oferece diversos níveis de linguagem, onde cada nível é um subconjunto do próximo nível. As mensagens apresentadas são referentes ao nível atual dos conceitos aprendidos pelos alunos até então. Podendo progredir até cinco níveis, iniciando em Beginner Student Language (BSL) até Advanced Student Language (ASL).

A metodologia definida para avaliação da efetividade das mensagens de erro partiu da premissa básica de como elas devem funcionar: se uma mensagem de erro é eficiente, é porque o estudante leu, conseguiu entender o seu significado, e pôde usar a informação transmitida para formular uma ação útil. Estudantes podem ficar presos em qualquer um destes passos, o objetivo, portanto foi avaliar o quão longe nestes passos o estudante consegue progredir em resposta a uma mensagem de erro (MARCEAU; FISLER; KRISHNAMURTHI, 2011). Para tal medição, foi criada uma tabela de rubricas para sinalizar as ações dos alunos, conforme mostra a Tabela 2.

Tabela 2 – Rubricas inferidas para cada edição de código.

Rubrica	**Descrição**
[DEL]	O estudante apagou o código problemático
[UNR]	O estudante faz algo que não tem relação ao erro e também não ajuda
[DIFF]	O estudante faz algo que não tem relação ao erro, mas corrige outro erro
[PART]	O estudante entendeu o erro e está no caminho de encontrar a solução
[FIX]	O estudante corrigiu o erro

Este trabalho ainda agrupou os erros em seis categorias: erros que correspondem a parênteses, índice fora dos limites, declarações, chamadas de função, condições e número de parâmetros. Realizando a medição efetiva das mensagens de acordo com estas categorias. Denominando "Efetividade" de acordo com a Equação 2.1:

$$p_{s,c} = \frac{FIX}{UNR + PART + FIX} \tag{2.1}$$

Como resultado do trabalho de Marceau, Fisler e Krishnamurthi (2011) foi observado que mensagens de erros de índice fora dos limites tiveram cerca de 84% de efetividade, assim como

erros relacionados a parênteses obtiveram cerca de 76%. Já as demais categorias de mensagens não se mostraram muito eficientes, especialmente na categoria de erros relacionados às chamadas de funções (36%).

É importante observar que existem diferenças entre o trabalho de Marceau, Fisler e Krishnamurthi (2011) e o que será realizado neste trabalho. No primeiro foi feita a análise dos erros sintáticos disparados pelo compilador, não por um AVA ou juiz *on-line*. Uma diferença bastante expressiva é que Marceau fez a separação das mensagens por categorias, não sendo interessante para este trabalho, visto que foram encontradas diversas mensagens de erros, o que implicaria em dicas muito genéricas devido a um grande quantitativo de erros encontrados da base analisada.

2.4.6 Portugol Studio

O Portugol Studio é um ambiente para auxiliar os alunos a programar em Portugol, que é uma pseudolinguagem que permite desenvolver algoritmos estruturados em português de forma simples e intuitiva. Possui uma sintaxe fácil, diversos exemplos e material de apoio à aprendizagem. Também possibilita a criação de jogos e outras aplicações. Atualmente o sistema conta com 72090 usuários cadastrados (STUDIO, 2017a).

O sistema é desenvolvido pela Universidade do Vale do Itajaí - SC e possui código-fonte aberto no GitHub[11] e diversos trabalhos acadêmicos já foram realizados utilizando-o.

Oferece suporte a leitura e escrita de arquivos, arquivos separados em abas, destaque de sintaxe com diferentes temas, console de entrada e saída de dados, localizar, substituir e exibição de mensagens de erro das análises feitas pelo núcleo.

Em (PELZ, 2014) foram realizadas diversas alterações no sistema com o objetivo de aprimorar o processo de correção automática de problemas. Uma das alterações efetuadas procurava identificar construções obrigatórias ou proibidas no código-fonte. A outra alteração foi na exibição de erros semânticos.

O trabalho ainda conta com a capacidade de comparação da solução do aluno com soluções modelo enviadas pelo professor. Para fazer esta comparação, as soluções enviadas pelo professor e a do aluno são normalizadas e então é aplicado o algoritmo de *Levenshtein* para descobrir o grau de semelhança entre elas (YUJIAN; BO, 2007).

A principal limitação do trabalho está na dificuldade de professores que não estão envolvidos no projeto conseguir criar problemas que utilizem os *Tree Walkers* - estratégia para percorrer os nós de uma árvore sintática abstrata de uma linguagem de programação - desenvolvidos porque o trabalho ainda possuía aspectos de prototipação. Não há especificação de quais mensagens de erros semânticos foram melhoradas. Além disso, o trabalho não englobou exercícios contendo laços de repetição e manipulação de vetores e matrizes.

[11] github.com

Outra limitação foi pela não realização de testes em exercícios que contenham muitas soluções modelo e não somente duas como realizado. Assim como a falta de experimentos com desenho experimental com alunos e professores, sob justificativa de calendário acadêmico. Uma outra limitação foi o fato de que esta ferramenta não funciona como juiz *on-line* ou ambiente virtual de aprendizagem, dificultando o acompanhamento individual dos aprendizes pelos docentes. Além do fato de ser somente em Portugol, que não é uma linguagem de programação propriamente dita.

2.5 Comparação dos trabalhos

A Tabela 3 ilustra uma comparação dos trabalhos encontrados de acordo com o *feedback*:

- Exibição dos casos de teste: se exibe os casos de testes onde ocorreu erro no código-fonte, obedecendo a dinâmica de juiz *on-line*;
- Possui mensagens melhoradas: se oferece o recurso de *feedback* melhorado dos erros sintáticos;
- É uma ferramenta *on-line*: se é uma ferramenta disponível *on-line* que pode ser acessada remotamente;
- Quais linguagens suporta: quais linguagens a ferramenta aplica seu estudo ou funcionalidade;
- Possui experimentação: se foi realizado alguma experimentação com o uso da ferramenta ou para validação de hipóteses.

O The Huxley aceita várias linguagens, mas, como resultado da implementação e integração da abordagem proposta por nosso trabalho no mesmo, oferece mensagens amigáveis (entendíveis) apenas em Python. Apesar disto, não há qualquer limitação ou restrição que impeça de se estender o suporte às outras linguagens disponíveis atualmente no The Huxley. Inclusive, já está em fase final de desenvolvimento o suporte à linguagem C, o que será melhor discutido na Seção 6.1.

Tabela 3 – Comparação dos trabalhos relacionados com o The Huxley.

Critério	**Expresso**	**CAP**	**InSTEP**	**Portugol**	**DrScheme**	**The Huxley + Proposta**
Exibe casos de teste				X		X
Mensagens melhoradas	X	X		X	X	X
On-line						X
Linguagem	Java	Pascal	C	Portugol	Scheme	Python, C, C++, Java e Octave
Experimentação			X	X	X	X

3

Especificação e Implantação da Solução

Neste capítulo são apresentados os processos necessários para desenvolvimento da ferramenta, abordando a especificação de requisitos, a implementação da solução e a integração com o The Huxley. Na primeira seção são apresentados o levantamento de requisitos e a elucidação. A segunda ilustra a implementação da ferramenta. Já a terceira e última, aborda a integração e funcionamento com o juiz *on-line* The Huxley.

3.1 Levantamento de Requisitos

Esta seção ilustra como foi obtida a base de dados através do acesso ao The Huxley e a elucidação dos requisitos funcionais.

3.1.1 Base de dados

A base de dados do The Huxley é relacional, mas permite o acesso à mesma através de uma API disponível através de Web Services que recebem requisições no formato JSON e retornam respostas também neste formato.

JSON[1]. É uma notação de objetos feita em JavaScript, é um formato criado para requisição de dados de forma leve. Provê uma notação de fácil leitura e escrita, tanto para seres humanos quanto para máquinas, ou seja, possui uma forma fácil para de analisar e gerar sua implementação. É completamente independente do idioma. O The Huxley utiliza esta notação na integração com sua plataforma web e para se acessar a base de dados.

As figuras 13, 15 e 14, mostram como é padronizada essa notação. Neste cenário, é criado um objeto com determinados atributos identificados por uma *string* a definir e posteriormente o seu valor.

[1] www.json.org

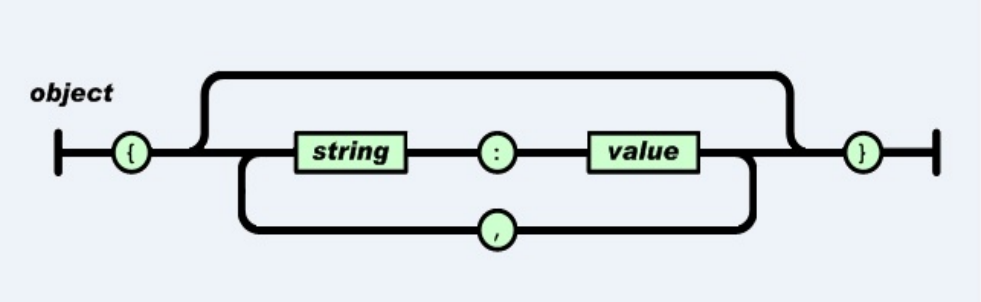

Figura 13 – Notação JSON para definição de um objeto[1].

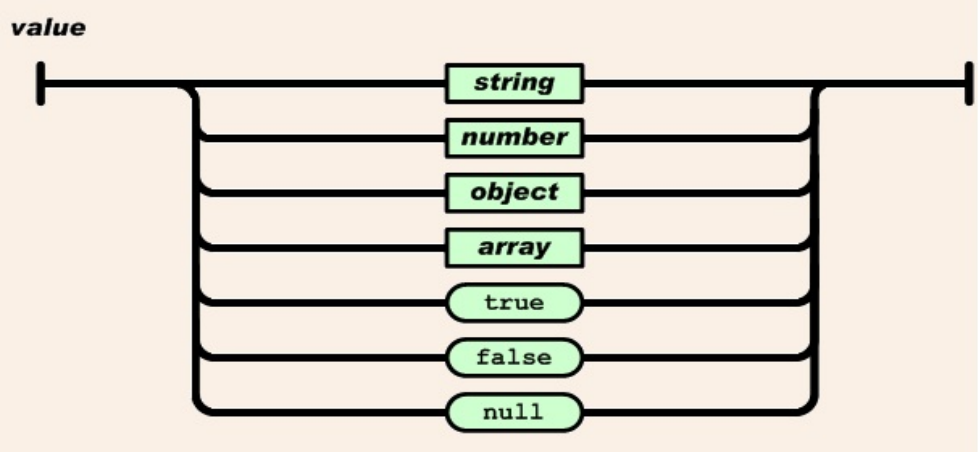

Figura 14 – Notação JSON dos tipos dos objetos, disponível em[1].

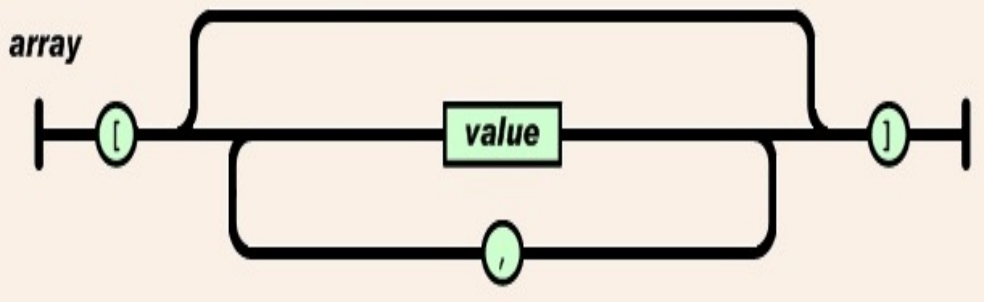

Figura 15 – Notação de lista de objetos JSON[1].

Um exemplo desta notação é mostrado no Código 3.1, onde é feita a especificação do objeto *"funcionarios"* e seus atributos, como primeiro nome (*"primeiroNome"*), sobrenome (*"sobrenome"*) e idade (*"idade"*), possuindo respectivamente valores atribuídos.

```
{"funcionarios":
  [ {"primeiroNome":"Marcio", "sobrenome":"Gois", "idade":"22"},
    {"primeiroNome":"Maria", "sobrenome":"Vieira", "idade":"23"},
    {"primeiroNome":"Arthur", "sobrenome":"Mota", "idade":"25"}
  ]
}
```

Listagem 3.1 – Exemplo de notação JSON.

Suas principais vantagens são: fácil manipulação, devido a facilidade de mapear as estruturas de dados homogêneas e heterogêneas disponíveis nas linguagens de programação; menor números de *bytes* na requisição, quando comparada às requisições com XML; separação de interesses da arquitetura e entre outras. A principal desvantagem é o esforço de se implantar um serviço de segurança, onde o tráfego de mensagens JSON deve ser criptografado antes de ser enviado, pois há o risco das mensagens serem interceptadas durante a comunicação (GONÇALVES; BASTOS; OLIVEIRA, 2014; ALEXANDER, 2008).

Como ilustrado na Seção 1.2, foram obtidas as estatísticas das submissões, assim como a base de dados com os erros das submissões realizadas na linguagem Python através de requisições feitas ao juiz *on-line* utilizando o padrão REST com JSON, como ilustra o arquivo localizado no caminho *Codes/baseErros.py* disponível em um compartilhamento do Google Drive[2].

A Figura 16 ilustra a notação da mensagem de erro mapeada, onde, inicialmente é identificada a linha que a contém, posteriormente sua classe e por fim, o subtipo do erro para então retornar uma mensagem de acordo com a abordagem desenvolvida.

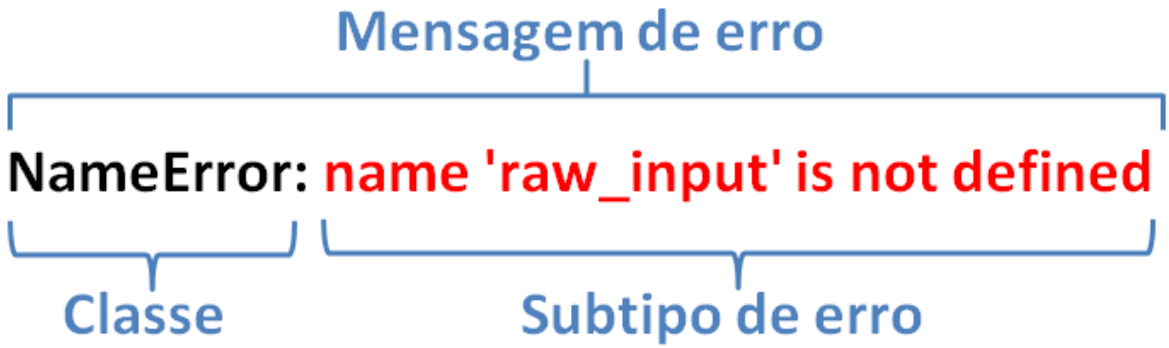

Figura 16 – Mensagem de erro mapeada.

Ao analisar a base, obtivemos 21 classes de erros, subdivididas em 144 mensagens de erros como ilustra a planilha disponível *on-line*[3]. A Figura 17 ilustra os nomes e sua frequência, ilustradas em ordem decrescente de ocorrência em sentido horário com suas respectivas cores.

[2] gg.gg/galileuFiles
[3] gg.gg/requi

Já a Tabela 4 ilustra as classes de erros com sua descrição e a quantidade de subtipos de erros mapeados.

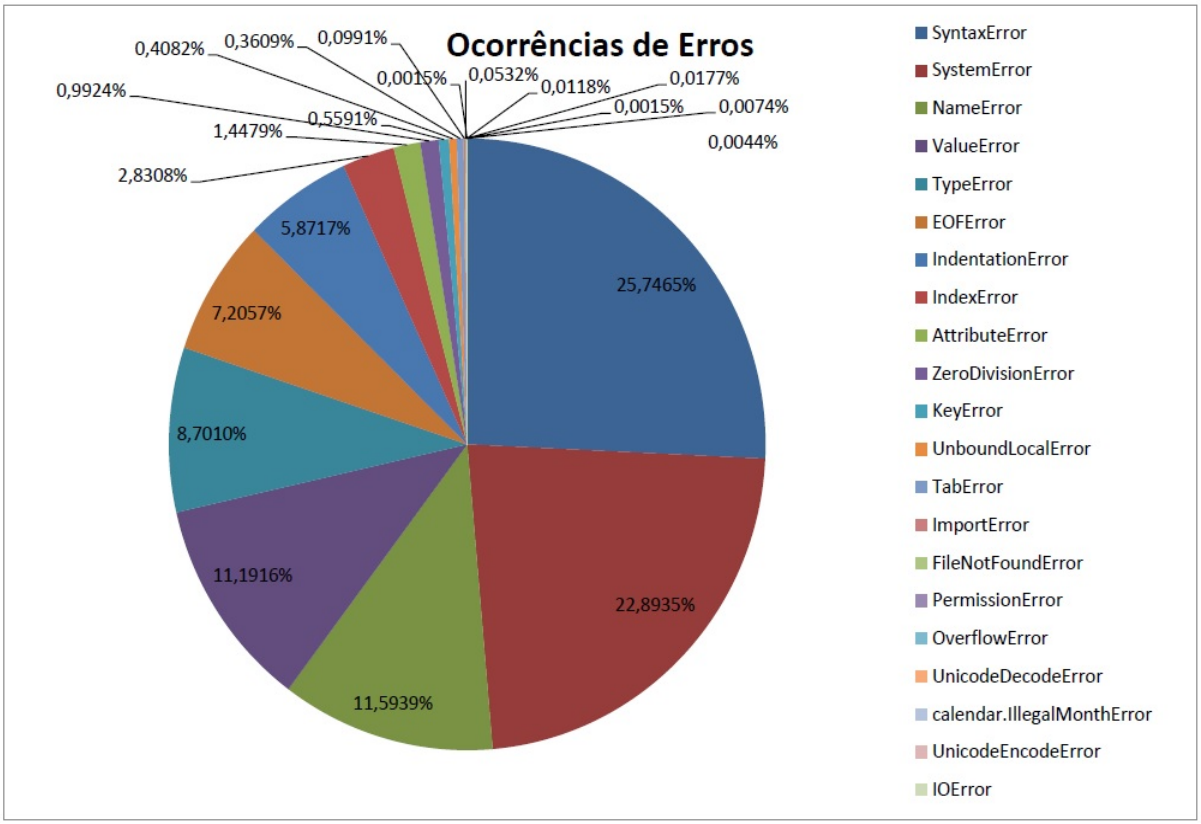

Figura 17 – Classes de erros da base do juiz *on-line* The Huxley.

Tabela 4 – Classes de erros.

Classe	**Descrição**	**Quantidade**
SyntaxError	São os erros referentes aos erros sintáticos, tais como uso incorreto de comandos, falta o uso incorreto de parâmetros ou atributos.	22
SystemError	São os erros referentes à execução do sistema. Geralmente algum diretório que não foi encontrado ou não foi possível abrir. Este erro foi eliminado do experimento, visto que não é erro referente ao código-fonte, mas sim ao sistema de execução.	1
NameError	São os erros referentes à nomeação de variáveis, métodos ou comandos que foram escritos de forma equivocada.	1
ValueError	São os erros referentes aos valores, seja de expressões ou tipos de dados.	27

Tabela 4 – Classes de erros.

Classe	**Descrição**	**Quantidade**
TypeError	São os erros referentes aos tipos dos dados, por atribuição, comparação ou expressão, com dados que não são do mesmo tipo.	64
EOFError	São os erros referentes a final de arquivo - *EOF (End Of File)*, onde acontece geralmente por uma leitura de dados feita de forma errada, onde a entrada não corresponde à leitura de dados realizada pelo código-fonte.	1
IndentationError	São os erros referentes a indentação, visto que Python é uma linguagem que possui delimitação de escopo por indentação.	5
IndexError	São os erros referentes aos índices de estruturas de dados, ao qual foi tentado acessar um determinado índice inexistente.	2
AttributeError	São os erros referentes aos atributos, acontece quando é feito uma tentativa de acesso a um atributo que não existe.	2
ZeroDivisionError	São os erros referentes a operações de divisões quando o divisor possui valor igual a zero.	4
KeyError	São os erros referentes a chaves, geralmente acontecem quando é feito o uso de uma chave que não existe para acessar uma determinada estrutura de dados.	1
UnboundLocalError	São os erros referentes ao uso de variáveis locais que não foram declaras, resolvidos, geralmente com sua declaração como global.	1
TabError	São os erros referentes ao uso do *Tab* para indentação, geralmente acontece quando há diferença entre seu uso para indentação.	2
ImportError	São erros referentes à importação de bibliotecas, geralmente acontece quando a biblioteca não existe ou não foi escrita da forma correta.	2
FileNotFoundError	São os erros referentes a caminhos em arquivos, quando o mesmo não existe ou não foi possível especificá-lo.	1

Tabela 4 – Classes de erros.

Classe	**Descrição**	**Quantidade**
PermissionError	São os erros referentes a permissões, quando há a tentativa de acessar um caminho em que o usuário não tem permissão para leitura ou escrita.	1
OverflowError	São os erros referentes ao uso de números que possuem um tamanho muito acima do permitido para um determinado tipo de dados.	3
UnicodeDecodeError	São os erros referentes ao uso de decodificação de caracteres não reconhecidos para o padrão utilizado, geralmente acontece quando se usa caracteres específicos de um determinado idioma ou símbolo.	1
calendar.IllegalMonthError	São os erros referentes ao uso de calendário, acontece quando o formato do mês, ano ou dia está incorreto.	1
UnicodeEncodeError	São os erros referentes ao uso de codificação de caracteres não reconhecidos para o padrão utilizado, geralmente acontece quando se usa caracteres específicos de um determinado idioma ou símbolo. A padronização mais comum é *utf8* ou *latin1*.	1
IOError	São os erros referentes à permissão para acessar um determinado diretório, geralmente acontece quando tenta gravar em arquivo em um local não permitido.	1

3.1.2 Requisitos Funcionais

Em engenharia de software, um requisito funcional define uma função de um sistema de software ou seu componente. Os requisitos funcionais podem ser cálculos, detalhes técnicos, manipulação de dados e de processamento e outras funcionalidades específicas que definem o que um sistema, idealmente, será capaz de realizar (PRESSMAN, 2016).

Ao analisar e catalogar os 144 subtipos de erros, foi feita a elucidação dos requisitos funcionais, através da análise da base de dados, assim como pesquisas do significado dos erros na internet e verificação dos códigos com os respectivos erros, disponível em[4]. Para cada subtipo de erro, que provém de um tipo de mensagem de erro, foi associado um requisito funcional que visa explicar o subtipo do erro.

A Tabela 5 ilustra os 21 (vinte e um) requisitos mais frequentes que, após o processo de análise, foram encontrados na base e distribuídos de acordo com a quantidade de problemas que foram utilizados na experimentação da ferramenta, representando cerca de 87.63% do total de ocorrência dos erros. Esta tabela também ilustra como é realizada a identificação do subtipo de erro a partir de padrões nas mensagens e o retorno de uma nova mensagem de acordo com cada mensagem de erro.

[4] gg.gg/galileuFiles

Tabela 5 – Requisitos Funcionais

Classe: subtipo de erro	**Requisito Funcional**	**Mensagem retornada**
SyntaxError: invalid syntax	RF0: Deve ser feita a implementação para identificar a palavra-chave "invalid syntax".	Verifique se está faltando algum parêntese, a ausência de ":"e declaração correta de alguns comandos, tais como: atribuição, if, for, while, input, print e outros.
NameError: name 'raw_input' is not defined	RF23: Identificar o erro pela palavra chave "is not defined".	Foi feito o uso de uma variável que não foi definida ou um comando que foi escrito de forma errada. Verifique a palavra 'X'.
ValueError: invalid literal for int() with base 10: '3 5 2'	RF24: Identificar o erro pela palavra-chave "invalid literal for". Pegar os tipos e valores.	O comando 'type' necessita de literais compatíveis com seu tipo. Verifique os tipos ou dados de leitura: valores.
EOFError: EOF when reading a line	RF115: Identificar o erro pela palavra-chave "EOF when reading a line".	EOF - end of file (final de arquivo) - ocorreu um erro de final de arquivo ao realizar a leitura dos dados. Verifique a declaração do input ou as entradas.
SyntaxError: Missing parentheses in call to 'print'	RF1: Identificar o erro utilizando a string "Missing parentheses in call to".	Verifique o uso do parêntese, a partir da versão 3.0 do Python seu uso é obrigatório no comando print.
IndexError: list index out of range	RF121: Identificar o erro pela palavra-chave "index out of range".	O índice da lista está fora do intervalo, verique o índice de acesso e seu valor.

Tabela 5 – Requisitos Funcionais

Classe: subtipo de erro	**Requisito Funcional**	**Mensagem retornada**
IndentationError: expected an indented block	RF116: Identificar o erro pela palavra-chave "expected an indented block".	Era esperado um bloco indentado. Por favor verifique a indentação. A indentação é uma característica peculiar na linguagem. Enquanto que os blocos são delimitados explicitamente em C, Java e PHP por chaves, em Pascal seria Begin e End e Fortran por palavras-chave como then e endif, em Python blocos são delimitados por espaços ou tabulações formando uma indentação visual. Não existem símbolos de “abre” e “fecha”. Fique atento para não errar na próxima!
IndentationError: unexpected indent	RF117: Identificar o erro pela palavra-chave "unexpected indent".	Indentação inesperada. Por favor verifique a indentação. A indentação é uma característica peculiar na linguagem. Enquanto que os blocos são delimitados explicitamente em C, Java e PHP por chaves e em Pascal e Fortran por palavras-chave como then e endif, em Python blocos são delimitados por espaços ou tabulações formando uma indentação visual. Não existem símbolos de "abre” e "fecha”. Fique atento para não errar na próxima!

Tabela 5 – Requisitos Funcionais

Classe: subtipo de erro	**Requisito Funcional**	**Mensagem retornada**
AttributeError: str' object has no attribute 'itmes'	RF123: Identificar o erro pela palavra-chave "object has no attribute". Pegar "type" e o nome do atributo "name".	O objeto do tipo "type" não tem nenhum atributo com o nome "name". Verifique se digitou corretamente o nome do atributo ou o objeto.
TypeError: unsupported operand type(s) for -: 'list' and 'list'	RF51: Identificar o erro pela palavra-chave "unsupported operand". Pegar o tipo e a operação.	A operação OP não é suportada para os tipos type e type. Verifique os tipos e suas operações.
IndentationError: unindent does not match any outer indentation level	RF118: Identificar o erro pela palavra-chave "unindent does not match".	A indentação não corresponde a nenhum nível de indentação externa, por favor verifique a indentação. A indentação é uma característica peculiar na linguagem. Enquanto que os blocos são delimitados explicitamente em C, Java e PHP por chaves e em Pascal e Fortran por palavras-chave como then e endif, em Python blocos são delimitados por espaços ou tabulações formando uma indentação visual. Não existem símbolos de "abre" e "fecha". Fique atento para não errar na próxima!
ValueError: could not convert string to float: '100 170 100 95'	RF25: Identificar o erro pela palavra-chave "could not convert string to float". Pegar os tipos e valores	Não é possível converter o tipo string para float, verique se os dados de leitura são do tipo float: 'valor'.

Tabela 5 – Requisitos Funcionais

Classe: subtipo de erro	**Requisito Funcional**	**Mensagem retornada**
ZeroDivisionError: float division by zero	RF127: Identificar o erro pela palavra-chave "division by zero".	Não é possível realizar divisão por zero. Verifique se isso ocorre em seu código.
TypeError: int' object is not iterable	RF52: Identificar o erro pela palavra-chave "object is not iterable". Pegar o 'type'.	O objeto do 'type' não é iterável, é apenas uma literal. Verifique se não é necessário o uso de uma lista ou o comando range(). Para transformar números em lista, você pode utilizar o operador [ ], exemplo: lista = [n1, n2, n3].
TypeError: unorderable types: str() <= int()	RF53: Identificar o erro pela palavra-chave "unorderable types".	Os tipos não são comparáveis, por isso não é possível realizar a comparação: types. Modifique-os para que sejam do mesmo tipo. Pegar os tipos depois dos dois pontos (:).
KeyError: '12' 12 1000 13 1 'a'	RF125: Identificar o erro pelo atributo "self.msg", ou seja, pela mensagem de erro, visto que não possui nenhuma palavra-chave que o identifique.	A chave digitada não é válida. Verifique seu tipo ou seu uso.
SyntaxError: unexpected EOF while parsing	RF3: Identificar pela palavra-chave "unexpected EOF while parsing".	Foram utilizados caracteres que não seguem o padrão da linguagem, assim ocorreu o erro de final de arquivo inesperado. Verifique os caracteres utilizados.

Tabela 5 – Requisitos Funcionais

Classe: subtipo de erro	**Requisito Funcional**	**Mensagem retornada**
TypeError: not all arguments converted during string formatting	RF55: Identificar o erro pela palavra-chave "not all arguments converted during string formatting".	Nem todos os argumentos foram convertidos durante a formatação. Verifique a formatação, os tipos das variáveis e argumentos.
TypeError: int() argument must be a string or a number, not 'list'	RF60: Identificar o erro pela palavra-chave "argument must be a string or a number". Pegar a função fun().	A função func() deve ter o argumento do tipo string ou um número, não outro tipo.
TypeError: Can't convert 'int' object to str implicitly	RF61: Identificar o erro pela palavra-chave "object to str implicitly". Pegar o 'type'.	Não é possível converter o tipo 'type' implicitamente para o tipo str, ou seja, por mais que seja compreensível sua conversão.
TypeError: int() argument must be a string, a bytes-like object or a number, not 'list'. Pegar 'type' que é list ou builtin_function_or_method. Pegar o tipo.	RF64: Identificar o erro pela palavra-chave "a bytes-like object or a number".	O argumento da função FUNC() deve ser uma sequência de caracteres ou bytes - como um objeto ou um número, não 'type'.

3.2 Implementação

Para implementação, foi utilizada a linguagem de programação Python, codificando usando a ferramenta de desenvolvimento Eclipse, na versão Neon.2. Buscou-se uma forma de implementação que pudesse atender a todos os requisitos e que fosse fácil inserir uma nova mensagem de erro. Para isto, foram utilizados padrões de projetos.

Segundo Vinoski (2002) em termos de uma implementação orientada a objetos, que utilize o padrão de projetos *Chain of Responsability* - Cadeia de Responsabilidade, tem-se o objetivo de desacoplar uma chamada de um objeto, ao interceptar uma cadeia de objetos. Isto permite que a cadeia de objetos atenda a uma solicitação de chamada à medida que encontra um determinado padrão na cadeia ou objeto. É muito utilizado em linhas de montagens e divisão de trabalhos.

Mapeando o problema como uma cadeia de responsabilidade, foi criada uma cadeia com as 21 (vinte e uma) classes de erros encontradas, onde cada uma tem subtipos de mensagens de erros, como ilustra a listagem 3.2, disponível no *main* do projeto. Ao realizar uma chamada à cadeia, passando a mensagem a ser melhorada, há a interceptação dos objetos para verificar se aquela mensagem está contida na cadeia. Caso esteja, é retornada a respectiva dica. Caso contrário, é informado que a mensagem está com erro de formação ou não foi encontrada.

```
1 chain = ['SyntaxE(self.msg).getErros()', 'SystemE(self.msg).getErros
    ()', 'NameE(self.msg).getErros()', 'ValueE(self.msg).getErros()',
    'TypeE(self.msg).getErros()', 'EOFE(self.msg).getErros()', '
    IndentationE(self.msg).getErros()', 'IndexE(self.msg).getErros()',
     'AttributeE(self.msg).getErros()', 'KeyE(self.msg).getErros()', '
    UnboundLocalE(self.msg).getErros()', 'FileNotFoundE(self.msg).
    getErros()', 'ZeroDivisionE(self.msg).getErros()', 'TabE(self.msg)
    .getErros()', 'ImportE(self.msg).getErros()', 'PermissionE(self.
    msg).getErros()', 'OverflowE(self.msg).getErros()', '
    UnicodeDecodeE(self.msg).getErros()', 'IllegalMonthE(self.msg).
    getErros()', 'UnicodeEncodeE(self.msg).getErros()', 'IOE(self.msg)
    .getErros()']
```

Listagem 3.2 – Código ilustrando a criação do padrão de projetos.

Cada classe de erro encontrada e ilustrada na seção anterior, foi mapeada como sendo uma classe em Python, possuindo suas respectivas mensagens de erros, identificação, atributos e métodos.

Foi criado um dicionário de dados, onde cada chave simboliza o erro sintático, ao realizar a busca, identifica-se o erro e então é chamado seu respectivo método para realizar o processamento da mensagem e então é retornada uma nova mensagem com a respectiva dica.

Com este mapeamento, ficou simples para adição de novas mensagens, uma vez que o

padrão de projeto e interceptação da cadeia não precisam ser modificados. Para adicionar uma nova classe de erro, basta criar uma nova classe e a inserir na cadeia do método *main* da listagem 3.2. Para inserir uma nova mensagem de erro, basta criar sua identificação no construtor da classe e o seu respectivo método que realiza o processamento, ao final, adiciona seu identificador na lista do dicionário de retorno do método *getErros()* da mesma. A Listagem 3.3 ilustra como é a especificação de uma nova classe e inserção de um novo erro.

```
1  class ClassName(object):
2
3      msg = ""
4      id = "ClassError" #id da classe de erro
5      keyError = {} #id do subtipo de erro
6
7      def __init__(self, msg):
8          self.msg = msg
9
10         self.keyError = {self.msg : self.keyErrorFunc()}
11
12     def methodName(self):
13         return """Mensagem processada"""+self.msg
14
15     #retorna um dicionario com o resultado do processamento
16     def getErros(self):
17         return {self.id:[self.keyError, self.keyError2]}
```

Listagem 3.3 – Código que ilustra a inserção de uma mensagem de erro.

O projeto segue a arquitetura de um *Web Service*, onde a classe *main* realiza sua execução bem como identificação da mensagem passada via requisição do cliente, criando a cadeia de responsabilidade e realizando a busca na mesma. Então, é feito o retorno ao cliente da mensagem com a respectiva dica, conforme ilustra a arquitetura da Figura 18.

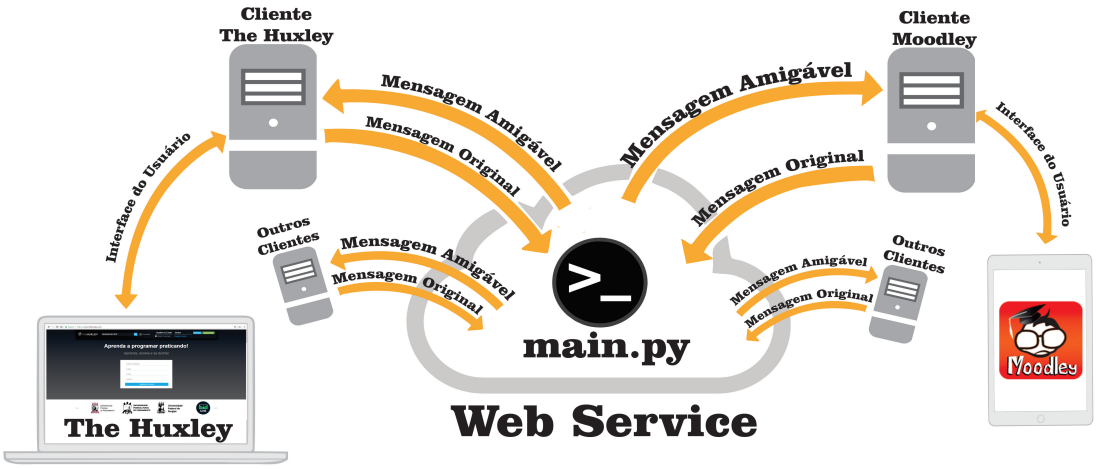

Figura 18 – Arquitetura da aplicação integrada ao The Huxley e ao Moodley.

Web service é uma solução utilizada na integração de sistemas e na comunicação entre aplicações diferentes, formado por um conjunto de nós de extremidades (cliente e servidor) que se comunicam através de trocas de mensagens com informações encapsuladas. Com esta tecnologia é possível que novas aplicações possam interagir com aquelas que já existem e que sistemas desenvolvidos em plataformas diferentes sejam compatíveis (CHRISTENSEN et al., 2001). A Figura 19 ilustra os passos realizados ao solicitar uma requisição ao *Web Service*, onde temos de um lado um cliente que faz uma requisição passando informações através de uma mensagem e do outro o servidor que responde a esta requisição.

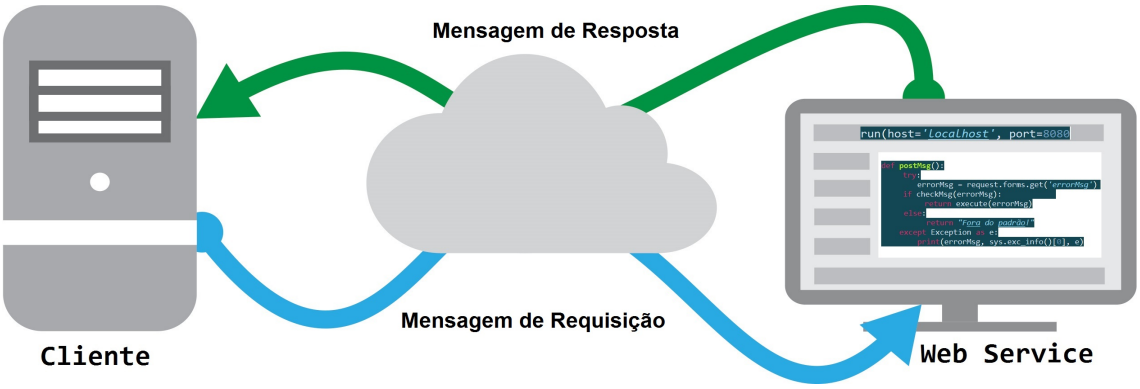

Figura 19 – Passos para realizar uma requisição ao *Web Service* (Imagem adaptada da internet).

Após a criação e especificação da arquitetura do código-fonte, foi necessário transformá-lo em um serviço web. Para isto foi utilizado o *framework web Bottle*[5]. A listagem 3.4 ilustra sua execução, onde o método *postMsg()* tenta capturar os parâmetros passados pelo usuário ao realizar uma determinada requisição ao servidor, onde é capturada a mensagem original do cliente contendo a mensagem de erro.

[5] bottlepy.org

```
@post('/')
def postMsg():
    try:
        errorMsg = request.forms.get('errorMsg')
        if checkMsg(errorMsg):
            return execute(errorMsg)
        else:
            print(errorMsg)
            return "Mensagem de erro fora do padrao!"
    except Exception as e:
        print(errorMsg, sys.exc_info()[0], e)

if __name__ == '__main__':
    debug(True)
    run(host='localhost', port=8080, debug=True, reloader=True)
```

Listagem 3.4 – Código exemplo executado na ferramenta Expresso.

Para maior estudo e verificação do código, o mesmo encontra-se disponível no caminho *Codes/MessageTH.rar*[6]. A subpasta */errors* contém a classe *main* para execução.

Para testar o projeto, é necessário executar o código e utilizar o PostMan ou criar uma requisição *POST* ao servidor local, como ilustra a Figura 20, onde:

(1) é a seleção do método *POST*;

(2) a *url* com a porta do servidor que está sendo executado;

(3) é o corpo da requisição, para edição de atributos;

(4) é o atributo *errorMsg*;

(5) é a mensagem de erro passada como parâmetro, correspondente ao atributo passado;

(6) é o retorno do processamento e execução do projeto, onde é retornada a mensagem com a dica de acordo com a requisição feita.

Também é possível enviar uma requisição através de um programa, como mostra a listagem 3.5 escrita em Python, possuindo os mesmos atributos da Figura 20. Esta requisição pode ser realizada por programas escritos em qualquer outra linguagem. Para que funcione, basta que os parâmetros estejam de acordo com a configuração do *web service*.

[6] gg.gg/galileuFiles

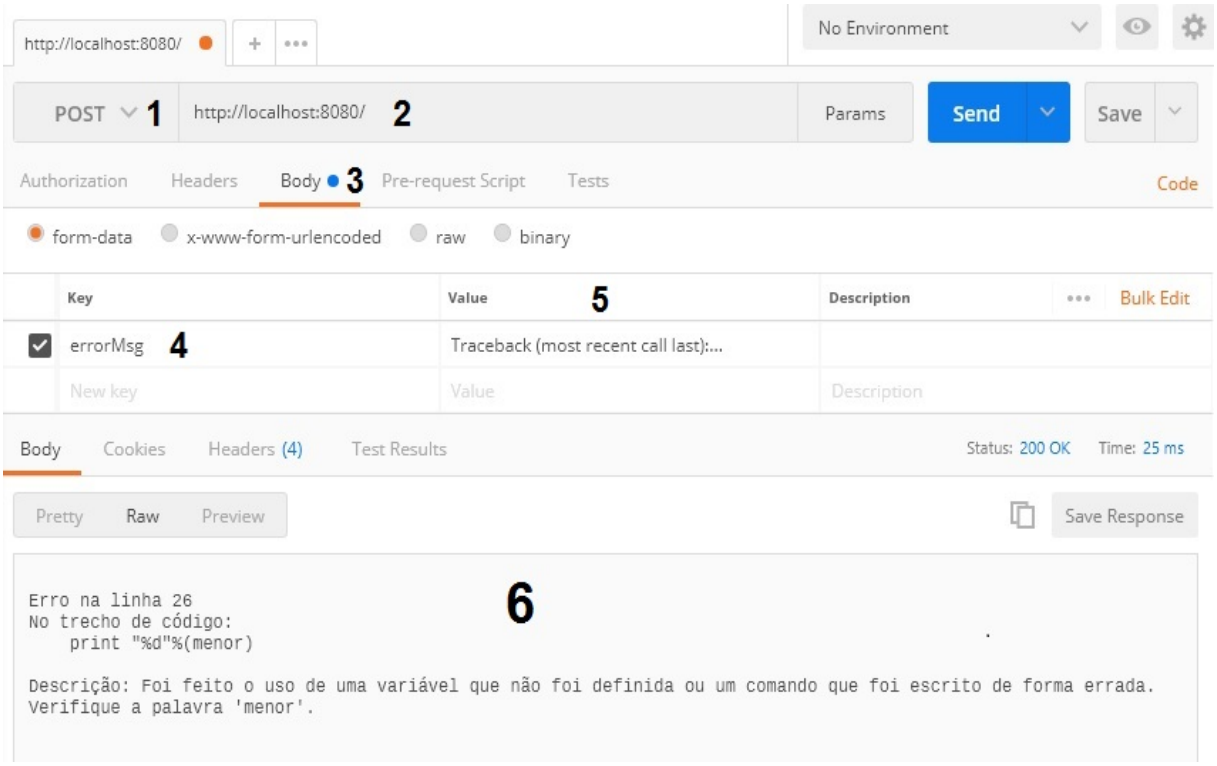

Figura 20 – Execução do Postman para testar o projeto.

```
1  import requests
2
3  url = "http://localhost:8080/"
4
5  payload = "------WebKitFormBoundary7MA4YWxkTrZu0gW\r\nContent-
       Disposition: form-data; name=\"errorMsg\"\r\n\r\nTraceback (most
       recent call last):\n
6  File \"/resp.py\", line 13, in <module>\n
7  elif ns%2==1:\n
8  TypeError: not all arguments converted during string formatting\n
9  Error in sys.excepthook:\n
10 Traceback (most recent call last):\n
11 File \"/usr/lib/python3/dist-packages/apport_python_hook.py\", line
       63, in apport_excepthook\n
12 from apport.fileutils import likely_packaged, get_recent_crashes\n
13 File \"/usr/lib/python3/dist-packages/apport/__init__.py\", line 5,
       in <module>\n
14 from apport.report import Report\n
15 File \"/usr/lib/python3/dist-packages/apport/report.py\", line 30, in
        <module>\n
16 import apport.fileutils\n
17 File \"/usr/lib/python3/dist-packages/apport/fileutils.py\", line 23,
```

```
    in <module>\n
18 from apport.packaging_impl import impl as packaging\n
19 File \"/usr/lib/python3/dist-packages/apport/packaging_impl.py\",
    line 20, in <module>\n
20 import apt\n
21 File \"/usr/lib/python3/dist-packages/apt/__init__.py\", line 34, in
    <module>\n
22 apt_pkg.init_config()\n
23 SystemError: E:Unable to read /etc/apt/apt.conf.d/ - opendir (13:
    Permission denied)\n\n
24
25 Original exception was:\n
26 Traceback (most recent call last):\n
27 File \"/resp.py\", line 13, in <module>\n
28 elif ns%2==1:\nTypeError: not all arguments converted during string
    formatting\n\r\n------WebKitFormBoundary7MA4YWxkTrZu0gW--"
29 headers = {
30     'content-type': "multipart/form-data; boundary=----
    WebKitFormBoundary7MA4YWxkTrZu0gW",
31     'cache-control': "no-cache",
32     'postman-token': "6fd76d11-d0cf-f894-4cff-22c8db7b94b3"
33     }
34
35 response = requests.request("POST", url, data=payload, headers=
    headers)
36
37 print(response.text)
```

Listagem 3.5 – Requisição POST na linguagem Python como exemplo de como testar o projeto.

3.3 Integração

A integração com o The Huxley foi feita através de interface gráfica como ilustram as figuras 21 a 24, onde ao clicar no link *ver saída amigável* é feita a requisição ao *web service* que está sendo executado, passando os respectivos parâmetros e é obtida uma *string* como retorno contendo o resultado do processamento da requisição. Esta parte, devido à necessidade de conhecer os detalhes de implementação da interface Web do The Huxley, foi realizada pela própria equipe de desenvolvedores do The Huxley.

As Figuras 21 e 22 ilustram a apresentação de mensagem erro informada ao usuário ao executar um determinado código-fonte que possui erro sintático.

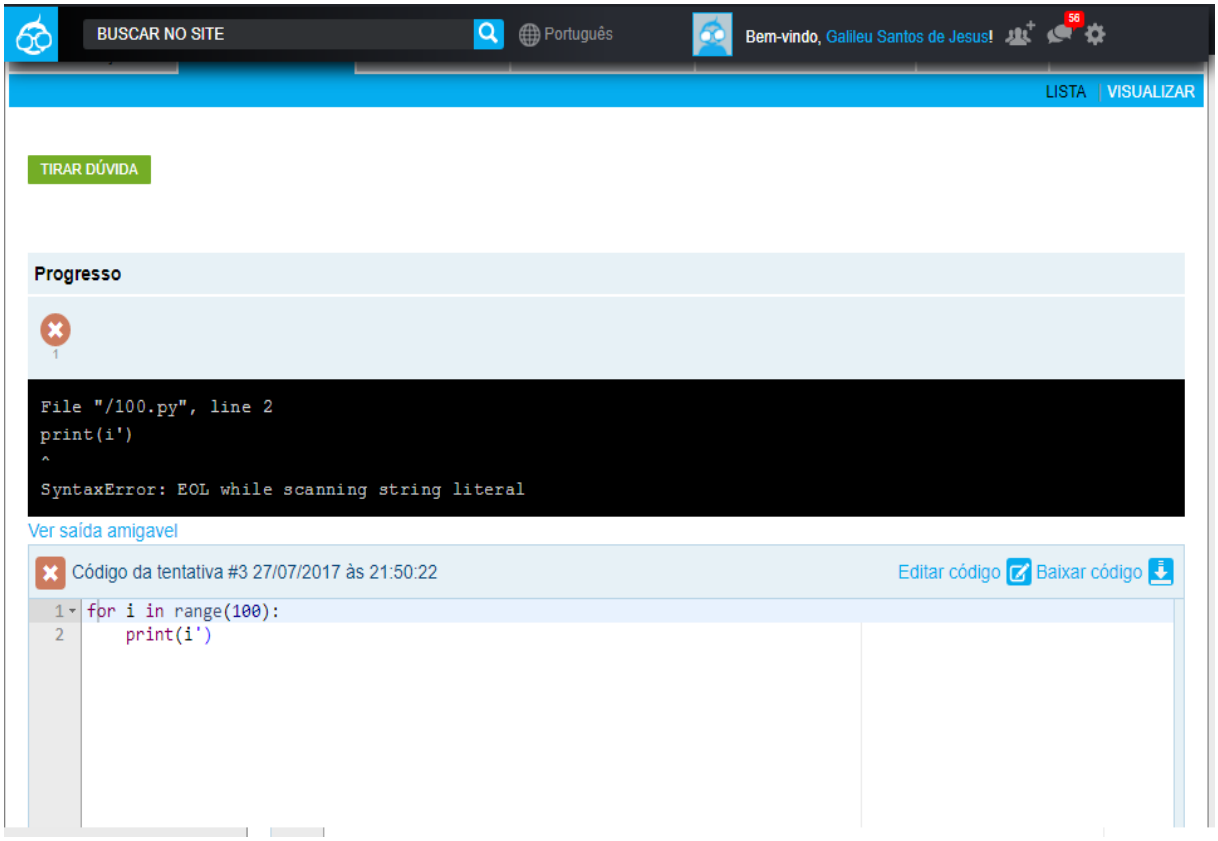

Figura 21 – Mensagem de erro Original, apresentada pelo The Huxley.

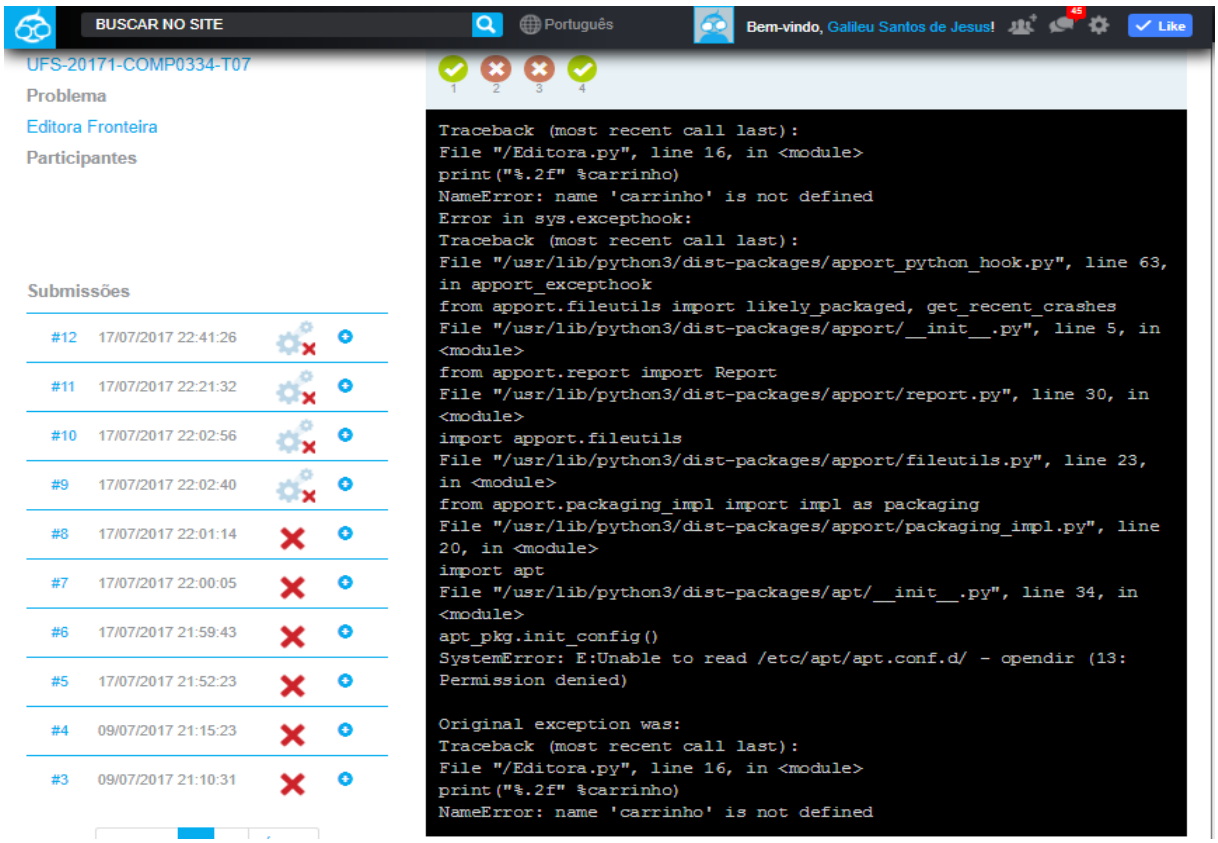

Figura 22 – Mensagem de erro Original, apresentada pelo The Huxley.

As Figuras 23 e 24 ilustram a apresentação de mensagem erro informada ao usuário ao executar um determinado código-fonte que possui erro sintático, ao clicar no link *ver saída amigável*, é realizado processamento e requisição à ferramenta desenvolvida, apresentando o resultado na tela do usuário.

Figura 23 – Mensagem de erro com saída amigável, apresentada pelo The Huxley após clicar em *ver saída amigável*.

Figura 24 – Mensagem de erro com saída amigável, apresentada pelo The Huxley após clicar em *ver saída amigável*.

Com a criação do *web service* é possível que novas aplicações possam interagir com o projeto desenvolvido através de requisições, sendo então realizada a integração com outro projeto de pesquisa, que tem o objetivo de desenvolver uma ferramenta para dispositivos móveis que possibilite o acesso ao The Huxley e ao Moodle. A Figura 25 ilustra o aplicativo móvel Moodley - criado para dispositivos da plataforma Android - integrado ao projeto desenvolvido por este trabalho, através da realização de requisição ao servidor criado.

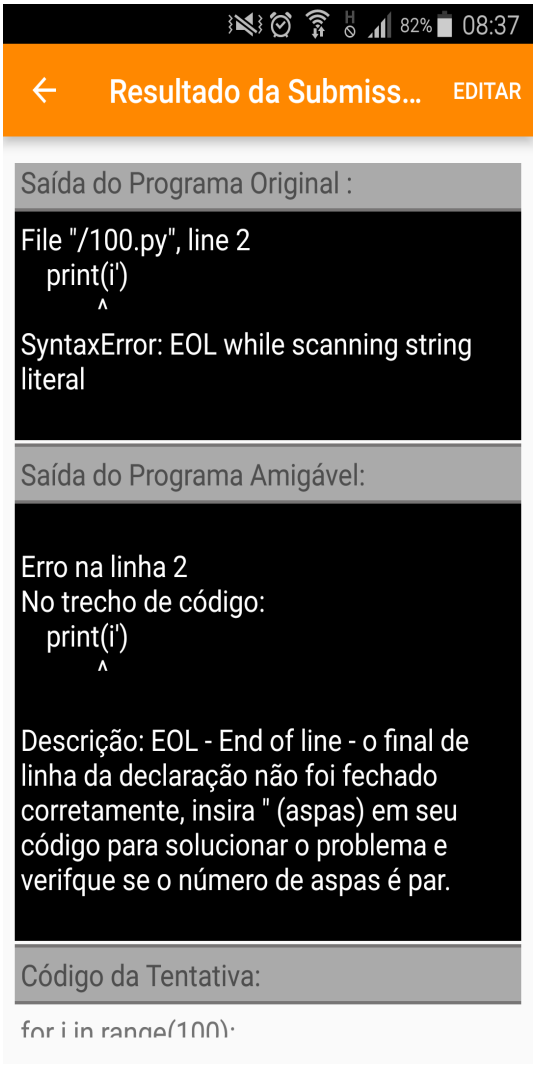

Figura 25 – Mensagem de erro com saída amigável apresentada pelo aplicativo móvel Moodley.

4

Estudo Experimental

Neste capítulo são apresentados os conceitos referentes ao estudo experimental, assim como toda a configuração do experimento tais como definição do experimento, definição do objetivo, planejamento do experimento, distribuição dos participantes, operação do experimento e por fim, análise e interpretação dos dados.

Diante do levantamento e análise descritos na Seção 3.1 sobre as mensagens de erros apresentadas no juiz *on-line* The Huxley, sistema alvo de estudo deste trabalho, pretende-se avaliar a utilização da ferramenta desenvolvida com base nos requisitos supracitados e a abordagem atual, onde denominamos esta última de mensagem original e a primeira de mensagem amigável.

Todavia, o estudo experimental deste trabalho tem por objetivo obter dados quantitativos relacionados à aplicação das duas abordagens: uso de mensagens amigáveis e o uso de mensagens originais, no processo de ensino-aprendizagem no contexto de alunos iniciantes em disciplinas de programação.

Nas seções seguintes são introduzidos os conceitos relacionados às fases do processo de experimentação segundo a orientação de (WOHLIN et al., 2000) para configuração do estudo experimental proposto neste trabalho.

4.1 Definição do Experimento

De acordo com Wohlin et al. (2000), para definir um estudo experimental, é necessário definir o objeto de estudo, o propósito, o foco quantitativo e qualitativo, a perspectiva e o contexto.

O objetivo deste estudo é avaliar, em um contexto real e por meio de experimento controlado, o *feedback* original fornecido pela ferramenta The Huxley e a abordagem amigável,

alvo de estudo deste trabalho.

Seguindo as orientações de Basili e Weiss (1984) ao utilizar o modelo GQM - do inglês, Goal Question Metric, tem-se que o objetivo foi: **Analisar** a abordagem atual de apresentação de mensagens de erros sintáticos, fornecida pelo juiz *on-line* The Huxley, **com a finalidade de** avaliar, **com respeito à** eficiência e eficácia no auxílio à correção de erros sintáticos de programas, **do ponto de vista** de programadores iniciantes, **no contexto de** disciplinas iniciais de programação.

4.2 Planejamento do Experimento

Nesta fase o planejamento do experimento é preparado para sua condução em que são definidos o contexto, as hipóteses, as variáveis, os participantes, o projeto do experimento, a instrumentação e por fim a avaliação de ameaças ao estudo.

O experimento foi realizado em turmas reais de graduação durante a disciplina Programação Imperativa, ofertada pelo Departamento de Computação da Universidade Federal de Sergipe.

Variáveis

De acordo com Wohlin et al. (2000) a definição das variáveis dependentes e independentes é de extrema importância para condução de um experimento controlado. As variáveis independentes são as variáveis de entrada do processo de experimentação e podem impactar sobre os resultados obtidos durante o experimento. As variáveis dependentes representam as variáveis de saída do processo de experimentação, ou seja, são avaliadas de acordo com os resultados obtidos da manipulação das variáveis independentes.

Para a especificação das hipóteses, serão consideradas as variáveis dependentes e independentes ilustradas na Figura 26.

Figura 26 – Variáveis dependentes e independentes para realização do experimento.

Hipóteses

As hipóteses definem o comportamento de acordo com o interesse da pesquisa, baseando-se em uma teoria ou suposição. A hipótese nula (H_0) é a de validação do experimento, já a alternativa (H_1) representa geralmente a hipótese da pesquisa em favor da qual a hipótese nula seja rejeitada (WOHLIN et al., 2000).

Para guiar o estudo e realizar o experimento, foram elaboradas quatro questões de pesquisa, cujas respostas buscam cumprir o objetivo de estudo do trabalho.

- Questão 1: Qual é o percentual de soluções que não foram aceitas, de acordo com a base de dados de erros do The Huxley?
- Questão 2: Como o novo *feedback* de mensagens foi recebido pelos alunos?
- Questão 3: As novas mensagens realmente ajudaram na correção dos erros?
- Questão 4: A ferramenta abrange os erros mais comuns da linguagem Python?

Serão consideradas as seguintes hipóteses, onde as hipóteses 1 (um) a 3 (três) se referem à métrica média de tempo e as demais à quantidade de erros corrigidos:

HIPÓTESE 1

Esta hipótese faz referência à comparação das duas abordagem em relação ao tempo, onde a média do mesmo está agrupada por participante do experimento.

Para validação desta hipótese, serão considerados apenas os alunos que obtiverem média final suficiente para lograr êxito na disciplina inicial de programação, visto que os alunos abaixo da média podem influenciar negativamente as duas abordagens, já que um aluno que não conseguiu êxito na disciplina, muito provavelmente não conseguirá entender as mensagens, pois possui dificuldades não para entendê-las, mas sim para compreender e implementar os códigos-fontes referentes à solução dos problemas.

- H_0 A abordagem original e a amigável têm a mesma eficiência referente à solução dos erros de programação.

(tempoOriginal = tempoAmigavel), onde tempoX representa a média de tempo de cada abordagem ao agrupar por participante.

- H_1 A abordagem original e a amigável não têm a mesma eficiência referente à solução dos erros de programação.

(tempoOriginal $\neq$ tempoAmigavel), onde tempoX representa a média de tempo de cada abordagem ao agrupar por participante.

HIPÓTESE 2

Esta hipótese faz referência à comparação das duas abordagem em relação ao tempo, onde a média do mesmo está agrupada por nível de inglês dos participantes do experimento.

- H_0 A abordagem original e a amigável têm a mesma eficiência referente à solução dos erros de programação.

(tempoOriginal = tempoAmigavel), onde tempoX representa a média de tempo de cada abordagem ao agrupar os participantes por nível de inglês.

- H_1 A abordagem original e a amigável não têm a mesma eficiência referente à solução dos erros de programação.

(tempoOriginal $\neq$ tempoAmigavel), onde tempoX representa a média de tempo de cada abordagem ao agrupar os participantes por nível de inglês.

HIPÓTESE 3

Esta hipótese faz referência à comparação das duas abordagens em relação ao tempo, onde a média do mesmo está agrupada por média com relação às notas na disciplina.

- H_0 A abordagem original e a amigável têm a mesma eficiência referente à solução dos erros de programação.

(tempoOriginal = tempoAmigavel), onde tempoX representa a média de tempo de cada abordagem ao agrupar por média de notas na disciplina.

- H_1 A abordagem original e a amigável não têm a mesma eficiência referente à solução dos erros de programação.

(tempoOriginal $\neq$ tempoAmigavel), onde tempoX representa a média de tempo de cada abordagem ao agrupar por média de notas na disciplina.

HIPÓTESE 4

Esta hipótese faz referência à comparação das duas abordagem em relação ao número de erros resolvidos por participante.

- H_0 A abordagem original e a amigável têm a mesma eficiência referente à solução dos erros de programação.

(errosOriginal = errosAmigavel), onde errosX representa a quantidade de erros resolvidos agrupados por participante.

- H_1 A abordagem original e a amigável não têm a mesma eficiência referente à solução dos erros de programação.

(errosOriginal = errosAmigavel), onde errosX representa a quantidade de erros resolvidos agrupados por participante.

HIPÓTESE 5

Esta hipótese faz referência à comparação das duas abordagens em relação ao número de erros resolvidos agrupados por nível de inglês dos participantes.

- H_0 A abordagem original e a amigável têm a mesma eficiência referente à solução dos erros de programação.

(errosOriginal = errosAmigavel), onde errosX representa a quantidade de erros resolvidos agrupados por nível de inglês.

- H_1 A abordagem original e a amigável não têm a mesma eficiência referente à solução dos erros de programação.

(errosOriginal = errosAmigavel), onde errosX representa a quantidade de erros resolvidos agrupados por nível de inglês.

HIPÓTESE 6

Esta hipótese faz referência à comparação das duas abordagem em relação ao número de erros resolvidos agrupados por média dos alunos na disciplina.

- H_0 A abordagem original e a amigável têm a mesma eficiência referente à solução dos erros de programação.

(errosOriginal = errosAmigavel), onde errosX representa a quantidade de erros resolvidos agrupados por média dos alunos na disciplina.

- H_1 A abordagem original e a amigável não têm a mesma eficiência referente à solução dos erros de programação..

(errosOriginal = errosAmigavel), onde errosX representa a quantidade de erros resolvidos agrupados por média dos alunos na disciplina.

Para todas as hipóteses, a H_0 é a hipótese nula, ao qual se deseja rejeitar, e a hipótese alternativa H_1 a que se deseja não rejeitar. São consideradas as variáveis dependentes e independentes do objeto de estudo do experimento.

Ressaltando que os termos eficiência e eficácia referenciados durante o trabalho estão relacionados ao tempo gasto para resolver um determinado erro sintático de um problema e a corrigir o erro, respectivamente.

Seleção dos participantes

Ao selecionar os participantes, estamos representando uma amostra da população, ou seja, são os indivíduos selecionados da população sob interesse para conduzir o experimento (WOHLIN et al., 2000).

A seleção dos participantes ocorreu de forma aleatória entre os que estavam presentes em horário da disciplina. Ao sortear um participante, foi solicitado que o mesmo participasse do experimento, confirmando que estaria disposto a cumprir as atividades e que teria tempo disponível para tal, visto que o experimento poderia ultrapassar o horário da aula. Foram selecionados 26 (vinte e seis) participantes, sendo possível somente a participação destes, por conta da logística e quantidade de máquinas disponíveis. Para acompanhamento do experimento, tivemos a ajuda de 4 (quatro) professores e 1 (um) monitor, alocados em três laboratórios.

Distribuição dos Participantes

O experimento foi projetado em um contexto pareado em que um mesmo grupo avaliará a abordagem original e outro a amigável, com a finalidade de responder as perguntas de pesquisa levantadas neste estudo.

Sendo assim, foi definido que cada participante resolveria 6 (seis) problemas, cada problema com 3 (três) erros sintáticos, onde seria feito o uso da abordagem amigável em 3 (três) problemas e a original nos outros 3 (três), representando os erros mais comuns ocorridos na base de dados do The Huxley. Foi usado um problema extra para que os participantes pudessem absorver os procedimentos a serem seguidos durante o experimento. Isso mitiga problemas relacionados ao tempo necessário para entender o fluxo de como se comportar durante o experimento.

A distribuição se deu da forma mais aleatória possível, sendo feita com o auxílio do programa em Python contido na Listagem 4.1, onde o *n-ésimo* participante representa o oposto do seu sucessor, sendo agrupados em pares, ou seja, o participante 1 (um) começa o problema 2 (dois) com a amigável, então o participante 2 (dois) começa com a abordagem original, como ilustra a ordem apresentada na Tabela 6, onde além do exposto, a sequência de solução de problemas também foi aleatória, para que não influenciasse no resultado final, simulando ao máximo situações cotidianas.

```
1 import random
2
3 n = int(input("Numero de participantes: "))
4
5 cont = 1
```

```
6
7  for i in range(n):
8          options1 = []
9          options2 = []
10         lrandom = [1, 1, 1, 0, 0, 0]
11         random.shuffle(lrandom)
12         for j in range(6):
13                 if lrandom[j]==1:
14                         options1.append(str(j+1)+' - Amigavel')
15                         options2.append(str(j+1)+' - Original')
16                 else:
17                         options1.append(str(j+1)+' - Original')
18                         options2.append(str(j+1)+' - Amigavel')
19         random.shuffle(options1)
20         random.shuffle(options2)
21         print("Participante", cont, ":", [j for j  in options1])
22         cont+=1
23         print("Participante", cont, ":", [j for j  in options2])
24         cont+=1
```

Listagem 4.1 – Código para distribuição dos participantes e ordem de realização do experimento.

Tabela 6 – Ordem dos participantes e escolha da abordagem. Ordem (Problema - Abordagem), onde A = Amigável e O = Original.

Participante	**Ordem (Problema - Abordagem)**
1	3 - A; 6 - O; 2 - O; 4 - A; 5 - A; 1 - O
2	3 - O; 1 - A; 6 - A; 5 - O; 2 - A; 4 - O
3	4 - O; 1 - O; 2 - A; 6 - A; 3 - O; 5 - A
4	2 - O; 4 - A; 6 - O; 1 - A; 3 - A; 5 - O
5	3 - O; 6 - O; 2 - O; 5 - A; 4 - A; 1 - A
6	6 - A; 4 - O; 1 - O; 3 - A; 2 - A; 5 - O
7	4 - A; 1 - O; 2 - A; 3 - O; 5 - A; 6 - O
8	2 - O; 3 - A; 6 - A; 5 - O; 1 - A; 4 - O
9	3 - O; 2 - A; 4 - A; 5 - O; 6 - A; 1 - O
10	3 - A; 2 - O; 4 - O; 5 - A; 6 - O; 1 - A
11	1 - O; 6 - A; 5 - O; 4 - A; 2 - O; 3 - A
12	4 - O; 2 - A; 1 - A; 5 - A; 6 - O; 3 - O
13	1 - A; 5 - O; 3 - O; 4 - A; 2 - O; 6 - A
14	4 - O; 6 - O; 2 - A; 5 - A; 3 - A; 1 - O

Tabela 6 – Ordem dos participantes e escolha da abordagem. Ordem (Problema - Abordagem), onde A = Amigável e O = Original.

Participante	**Ordem (Problema - Abordagem)**
15	4 - A; 1 - O; 2 - O; 3 - O; 6 - A; 5 - A
16	4 - O; 2 - A; 5 - O; 1 - A; 6 - O; 3 - A
17	2 - A; 3 - O; 1 - O; 5 - A; 4 - A; 6 - O
18	3 - A; 4 - O; 1 - A; 6 - A; 2 - O; 5 - O
19	6 - O; 5 - O; 3 - A; 4 - A; 2 - A; 1 - O
20	2 - O; 5 - A; 1 - A; 6 - A; 3 - O; 4 - O
21	6 - O; 1 - O; 5 - A; 2 - A; 3 - A; 4 - O
22	1 - A; 2 - O; 6 - A; 3 - O; 5 - O; 4 - A
23	6 - O; 3 - A; 2 - A; 4 - O; 1 - A; 5 - O
24	5 - A; 4 - A; 6 - A; 3 - O; 2 - O; 1 - O
25	6 - O; 5 - A; 4 - A; 2 - O; 1 - O; 3 - A
26	5 - O; 4 - O; 2 - A; 1 - A; 3 - O; 6 - A

Ao final do experimento, foi realizado um questionário para podermos responder algumas questões de pesquisa e sabermos opiniões dos participantes acerca das abordagens, assim como informações qualitativas sobre os mesmos, como ilustra o Apêndice D.

Escolha dos problemas

Os problemas foram selecionados a partir de uma busca feita nas submissões que continham erros sintáticos na base do The Huxley. Ao encontrar um determinado código-fonte que apresentava aquele erro, foi verificado o nível do problema e se sua descrição estava coesa. A seguir são ilustrados os problemas de forma geral.

(A) Problema Exemplo: Mínimo múltiplo comum[1], nível 2
Dados dois números inteiros positivos, calcular o mínimo múltiplo comum. A entrada consiste dos números inteiros *m* e *n*, ambos maiores que zero;

(B) Problema 1: Elevador Seguro[2], nível 2
Genival é ascensorista da Prefeitura Municipal e todos os dias conduz os passageiros para cima e para baixo. O elevador que ele opera tem capacidade para no máximo 7 pessoas e no máximo 560 quilos.
Escreva um programa que receba como entrada o peso de várias pessoas que estão entrando

[1] www.thehuxley.com/problem/842
[2] www.thehuxley.com/problem/467

no elevador e exiba quantas poderão ser transportadas com segurança e o peso total carregado, respeitando os limites do elevador;

(C) Problema 2: Aprovados no Concurso[3], nível 2
O IBGE realizou um concurso para contratar pessoas para trabalhar no censo. Cada candidato fez uma prova de português com 50 questões, outra de matemática com 35 questões, e uma prova de redação.
Para ser aprovado, era necessário acertar pelo menos 80% da prova de português, 60% da prova de matemática, e ter nota igual ou superior a 7 na redação;
Escreva um programa que receba como entrada, para cada candidato, a quantidade de questões certas em português e em matemática, e também a nota na redação, e depois exiba quantos candidatos foram aprovados.

(D) Problema 3: Caracteres[4], nível 1
Você deve ler uma sequência de caracteres e imprimir na ordem inversa da leitura. A entrada consiste de um inteiro n, indicando quantos caracteres devem ser lidos e uma sequencia de n caracteres. A entrada termina quando $n=0$;

(E) Problema 4: Desafio do menor e maior número[5], nível 2
Leonardo é um garoto muito criativo. Ele adora criar desafios para seus colegas da escola. Seu último desafio é o seguinte: diversos números são ditos em voz alta, quando o número 0 (zero) é dito então o desafio termina e seus colegas devem dizer imediatamente qual foi o menor e o maior número. Leonardo tem muita dificuldade de verificar se a resposta dada pelos colegas é correta ou não, pois a sequência de números costuma ser longa.
Por este motivo, ele resolveu pedir sua ajuda.Sua tarefa é escrever um programa que dada uma sequência de números inteiros positivos terminada por 0 (zero), imprime o menor e o maior da sequência;

(F) Problema 5: Equação do Segundo Grau[6], nível 1
Toda vez que Ambrósio vai calcular as raízes de uma equação do segundo grau, esquece de algum detalhe e calcula errado. Para evitar esquecimentos, resolveu fazer um programa que calcula as raízes da equação de segundo grau. A entrada consiste dos números reais a, b e c, que correspondem aos coeficientes da equação de segundo grau ($ax^2+bx+c=0$);

(G) Problema 6: Balanceando números[7], nível 2
Balanceamento de parênteses é algo fundamental na computação e na matemática. Mas, e se no lugar de parênteses, tivéssemos números? Faça um programa que avalie se uma sequência de números está balanceada ou não.

[3] www.thehuxley.com/problem/473
[4] www.thehuxley.com/problem/9
[5] www.thehuxley.com/problem/956
[6] www.thehuxley.com/problem/6
[7] www.thehuxley.com/problem/1234

Sendo '(' representado por 0 e ')' representado por 1, a sequência: 0 0 1 1 está balanceada, assim como: 0 1 0 1 está balanceada.

Definição dos erros e códigos-fontes

Foi disponibilizado para cada problema um código-fonte apresentando 3 (erros) sintáticos. Os participantes têm o objetivo de corrigi-los, estabelecendo um tempo total de 10 (dez) minutos para resolver os erros apresentados. Ao ter acesso ao código-fonte, foi orientado que o participante submetesse de imediato, assim como no final, para que fosse possível realizar a medição do tempo de início e fim de cada problema a partir do registro do próprio The Huxley. A seguir são apresentados os códigos-fontes referentes ao problema exemplo, assim como os seus erros e o seu código solução onde ilustra os erros corrigidos. O restante dos códigos-fontes referentes aos problemas do experimento encontram-se no Apêndice A.

- **Problema Exemplo: Mínimo múltiplo comum**

A Listagem 4.2 ilustra o código-fonte para o problema exemplo, onde são apresentados 3 (três) erros sintáticos, são eles: #erro 1, linha 10 - TypeError: int() argument must be a string, a bytes-like object or a number, not 'list'; #erro 2, linha 10 - TypeError: Can't convert 'int' object to str implicitly; e #erro 3, linha 12 - SyntaxError: EOL while scanning string literal.

```
1  def MMC(a, b):
2      i = a*b
3      m = a*b
4      while(i>=a and i>=b):
5          if i%a == 0 and i%b == 0:
6              m = i
7          i = i-1
8      return m
9  nums = input().split()
10 a = int(nums[0].split(int())) #erro 1 e erro 2
11 b = int(nums[1])
12 print(MMC'(a.split(),b)) #erro 3
```

Listagem 4.2 – Código-fonte para o problema exemplo, apresentando 3 (três) erros sintáticos.

A Listagem 4.3 ilustra o código-fonte após a correção dos 3 (três) erros supracitados.

```
1  def MMC(a, b):
2      i = a*b
3      m = a*b
4      while(i>=a and i>=b):
5          if i%a == 0 and i%b == 0:
6              m = i
```

```
7           i = i-1
8       return m
9   nums = input().split()
10  a = int(nums[0])
11  b = int(nums[1])
12  print(MMC(a,b))
```

Listagem 4.3 – Código-fonte para o problema exemplo com os erros corrigidos.

4.3 Operação do Experimento

Antes do experimento ser executado, foram realizadas algumas tarefas referentes à preparação do experimento, expostas a seguir.

Alocação dos participantes

Todos os participantes foram alocados de acordo com sua numeração na lista de presença e sorteios supracitados, onde foi dado um tempo inicial para leitura dos problemas e logo após, foi iniciado o experimento.

Ambiente do Experimento

O ambiente configurado para execução do experimento foi o mesmo para todos os participantes, todos utilizaram o mesmo navegador *web*, *Google Chrome*, para acessar o juiz *online* The Huxley e o sistema operacional ArchLinux para manuseio dos arquivos do experimento. É importante ressaltar que todos os alunos participantes já tinham utilizado as máquinas do laboratório durante o semestre, portanto já estavam habituados ao ambiente do experimento.

Treinamento dos participantes

Todos os participantes receberam o mesmo treinamento, visto que foi o mesmo professor que lecionou nas turmas dos participantes, aplicando as mesmas atividades e exercícios.

Material para execução do experimento

Foi criado um site[8] para que todos os participantes tivessem acesso a todo o material sem nenhum transtorno e para que não atrasasse o experimento. Antes de iniciar, foi mostrado todo o material criado disponível no site, são eles: códigos-fontes com os erros dos problemas; material com os nomes dos problemas e o link de acesso para cada um; ordem dos participantes identificados por um número, dicas e ordem de execução do experimento e por fim, o formulário final de questões a serem respondidas ao finalizar o experimento.

Além disso, foi disponibilizado documento impresso com identificação de cada participante de acordo com o sorteio, com a sua respectiva ordem de realização do experimento, para

[8] gg.gg/Xpython

que não ocorresse nenhum transtorno. Para tal, os acompanhantes do experimento tiveram a missão de verificar se a ordem estabelecida foi cumprida rigorosamente pelos participantes.

Execução do experimento

Ao executar o experimento, os participantes recebiam o *feedback* de mensagens de acordo com a ordem estabelecida e entregue. As imagens localizadas no Apêndice B ilustram passo a passo a exibição das mensagens de erros em cada abordagem e em cada problema.

4.4 Coleta de Dados

Para categorizar se o participante resolveu todos os erros de um determinado problema, foi adotada a metodologia de forma simplificada do trabalho de Marceau, Fisler e Krishnamurthi (2011), conforme ilustra a Tabela 7. Esta notação foi também utilizada para agrupar os dados de acordo com o tipo de rubrica.

Tabela 7 – Rubricas utilizadas para o resultado das submissões do experimento.

Rubrica	**Descrição**
[FIX]	Corrigiu o erro
[UFIX]	Não corrigiu o erro
[UNR]	Fez algo que não tem relação ao erro e não corrigiu

Ao realizar o experimento, como métricas importantes a serem colhidas, foram obtidos além do tipo da abordagem utilizada e identificação do participante, o número de submissões realizadas, a quantidade de erros corrigidos, o tempo em segundos para resolução de um problema, o resultado de acordo com as rubricas supracitadas, o número do problema, onde: p1 = problema 1; p2 = problema 2; p3 = problema 3; p4 = problema 4; p5 = problema 5 e p6 = problema 6, o nível de inglês informado por cada participante e por fim, a média final obtida pelo aluno na disciplina de programação.

A Figura 27 ilustra a tela com os dados salvos no banco de dados do The Huxley, onde temos que:

(1) Representa a identificação do participante;

(2) Representa o problema para qual o participante realizou as submissões;

(3) Representa o *status* do problema: se foi aceito ou não. É marcado com este ícone quando o participante resolve todos os 3 (três) erros do problema. Quando não resolveu todos, foi feita uma análise no código para verificar quais erros foram resolvidos;

(4) Representa o número de submissões realizadas;

(5) O horário em que uma determinada submissão foi realizada

(6) e (7) Ilustram a primeira e a última submissão realizada, respectivamente, as quais foram utilizadas para mensurar o horário inicial e final para resolução de um problema;

(8) Representa os casos de testes aos quais o código-fonte é submetido.

Figura 27 – Tela do The Huxley com os dados do experimento salvo por participante.

Ao identificar todos os participantes, foi feita a coleta dos dados, conforme ilustrado na Tabela 18 localizada no Apêndice C, também disponível no documento compartilhado do Google Drive[9], no caminho */Coleta de Dados*.

Ao final do experimento, foi aplicado um questionário qualitativo para identificar opiniões e características dos participantes, para um possível agrupamento na análise do experimento. Sua formulação e análises estão ilustradas no Apêndice D.

[9] gg.gg/galileuFiles

5

Análise dos Resultados

A análise dos dados e a avaliação das hipóteses tiveram como embasamento a estatística descritiva, com a qual foram calculadas média, desvio padrão, quartil, coeficiente de variância além dos testes estatísticos para validação das hipóteses. Toda análise e geração de gráficos foi realizada utilizando a ferramenta R versão 1.0.153 e Microsoft Excel 2010. Todo o material de análise referente ao agrupamento dos resultados e uso de testes estatísticos estão disponíveis no documento compartilhado do Google Drive[1], localizado na pasta */Estatística*.

Primeiramente foi realizada a análise descritiva dos resultados do experimento, logo após, foi realizada a verificação da normalidade dos dados, em seguida, a avaliação das hipóteses considerando o tempo de implementação das soluções e a quantidade de erros corrigidos, definidos anteriormente. Por fim, são discutidas as ameaças à validade do estudo experimental.

5.1 Análise Descritiva dos Resultados

Na análise descritiva dos resultados, foi considerado todo o estudo realizado para obtenção dos dados, o experimento e o questionário final aplicado ao fim do mesmo.

Para responder a Questão de Pesquisa 1: Qual é o percentual de soluções que não foram aceitas, de acordo com a base de dados de erros do The Huxley? Foi feita uma análise na base de dados do The Huxley, como ilustrado na Seção 1.2 e na Figura 2, onde 70,30% das submissões realizadas são consideradas como erradas.

Para responder a Questão de Pesquisa 2: Como o novo *feedback* de mensagens foi recebido pelos alunos? Foi fornecido um questionário ao final do experimento para colher os dados referentes às opiniões dos participantes acerca da ferramenta fornecida. Suas respostas encontram-se no Apêndice D. No geral, todos os estudantes acharam importante que um juiz *on-line* possua mensagens amigáveis, informando que é de extrema importância para uma melhor

[1] gg.gg/galileuFiles

compreensão do erros, pois mesmo sabendo inglês às vezes a mensagem não é tão clara. Uma resposta bastante interessante que um participante informou foi que a ferramenta serve para "estimular o aluno a resolver as questões sem desistir na metade ou não compreender direito o erro ocorrido", isto reforça a proposta de diminuir o medo e frustração dos aprendizes em programação.

Para responder a Questão de Pesquisa 3: As novas mensagens realmente ajudaram na correção dos erros? Foram fornecidas perguntas específicas no questionário supracitado que abordem este tema. Considerando uma faixa com 5 (cinco) opções: 1-Muito Baixo; 2-Baixo; 3-Médio; 4-Alto; 5-Muito Alto. Foi obtido o resultado ilustrado na Figura 85 do Apêndice D, sendo que 7,7% dos participantes informaram que sua utilidade foi muito baixa, cerca de 23,1% sinalizam que possui média utilidade, cerca de 38,5% afirmam que tem uma utilidade alta na correção de erros e por fim, 30,6% informaram que é muito alta a utilidade de mensagens amigáveis.

Para responder a Questão de Pesquisa 4: A ferramenta abrange os erros mais comuns da linguagem Python? Foram feitas análises na base de erros pertencentes ao The Huxley, onde foi realizado o mapeamento de todos os erros encontrados na mesma. Apesar de abranger todos os erros da base encontrada, não podemos afirmar que abrange os erros mais comuns da linguagem Python, porém abrange praticamente a totalidade dos erros dos usuários desta ferramenta.

A Figura 28 representa o *boxplot* da média dos tempos em segundo das duas abordagens agrupada por participante, de acordo com a Hipótese 1. Os resultados indicam que o tempo total gasto pela abordagem amigável foi menor que a original, visto que a soma total da primeira foi de 25.816 segundos, enquanto que na segunda foi de 29.194, além de possuir uma mediana menor. O ponto localizado acima da abordagem original, sinaliza um *outlier*, ou seja, é o tempo máximo atingido por um participante ao utilizar determinada abordagem, conforme ilustrado na Tabela 8.

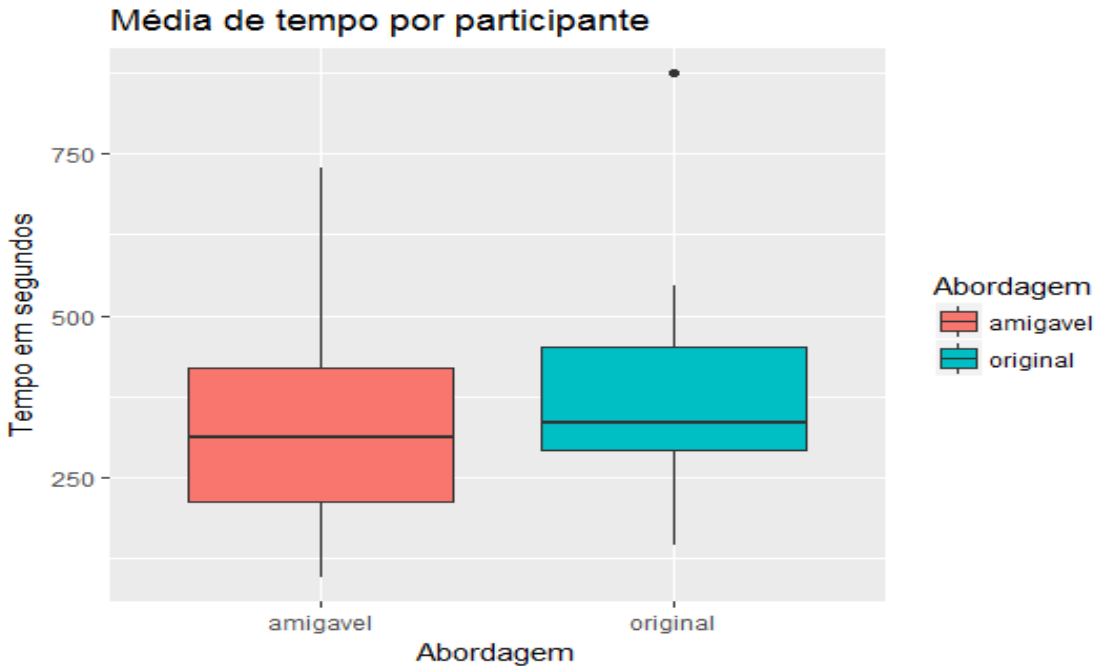

Figura 28 – *Boxplot* da média dos tempos das duas abordagens agrupada por participante.

Tabela 8 – Estatística descritiva da média de tempo agrupada por participante.

Abordagem	**Mínimo**	**1^o quartil**	**Mediana**	**Média**	**3^o quartil**	**Máximo**
Amigável	96,0	213,5	311,5	331,0	420,1	727,3
Original	145,3	293,2	335,5	374,3	452,6	874,7

A Figura 29 representa o *boxplot* da média dos tempos das duas abordagens agrupada por nível de inglês, de acordo com a Hipótese 2. A figura ilustra uma diferença na mediana dos dados, também a média da abordagem amigável foi menor, 330,5 contra 367,4, assim como o ponto máximo, primeiro quartil e terceiro quartil. Porém os pontos mínimos se mostraram bem próximos, conforme ilustra a Tabela 9.

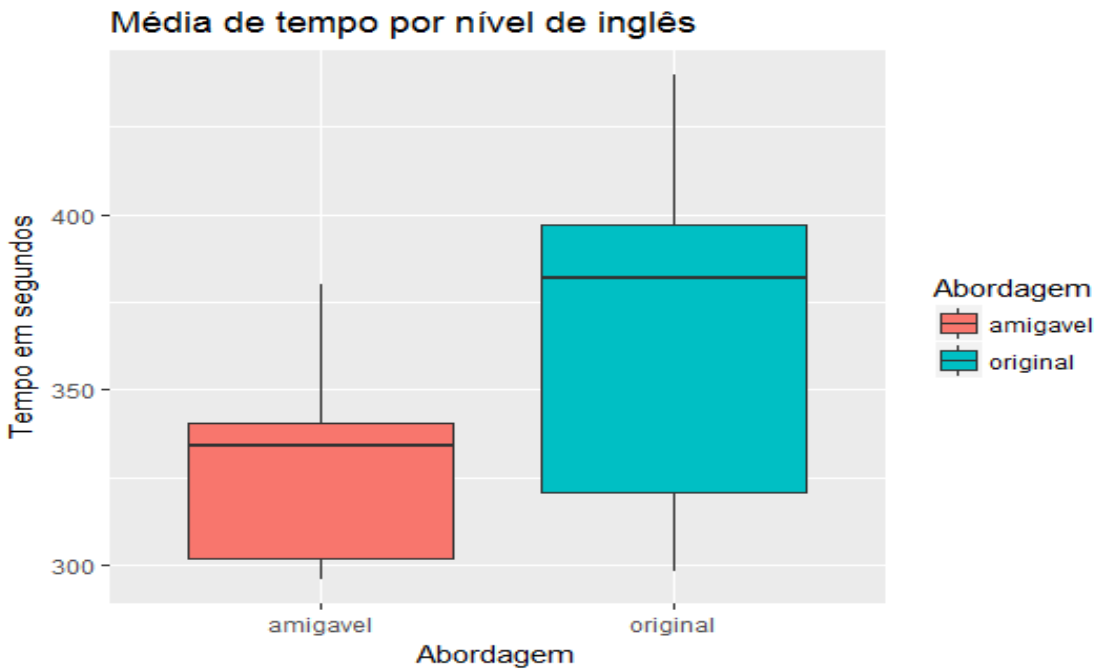

Figura 29 – *Boxplot* da média dos tempos das duas abordagens agrupada por nível de inglês.

Tabela 9 – Estatística descritiva da média de tempo agrupada por nível de inglês.

Abordagem	**Mínimo**	**1^o quartil**	**Mediana**	**Média**	**3^o quartil**	**Máximo**
Amigável	295,9	301,9	334,1	330,5	340,7	380,1
Original	298,1	320,6	381,7	367,4	396,9	439,8

A Figura 30 representa o *boxplot* da média dos tempos em segundos das duas abordagens agrupadas pela média final obtida pelos alunos na disciplina, de acordo com a Hipótese 3. A figura ilustra uma diferença na mediana dos dados, também a média da abordagem amigável foi menor, 292,6 contra 388,8, assim como o ponto máximo e mínimo, primeiro quartil e terceiro quartil. Ambas as abordagens mostraram possuir *outliers*, conforme ilustrado na Tabela 10.

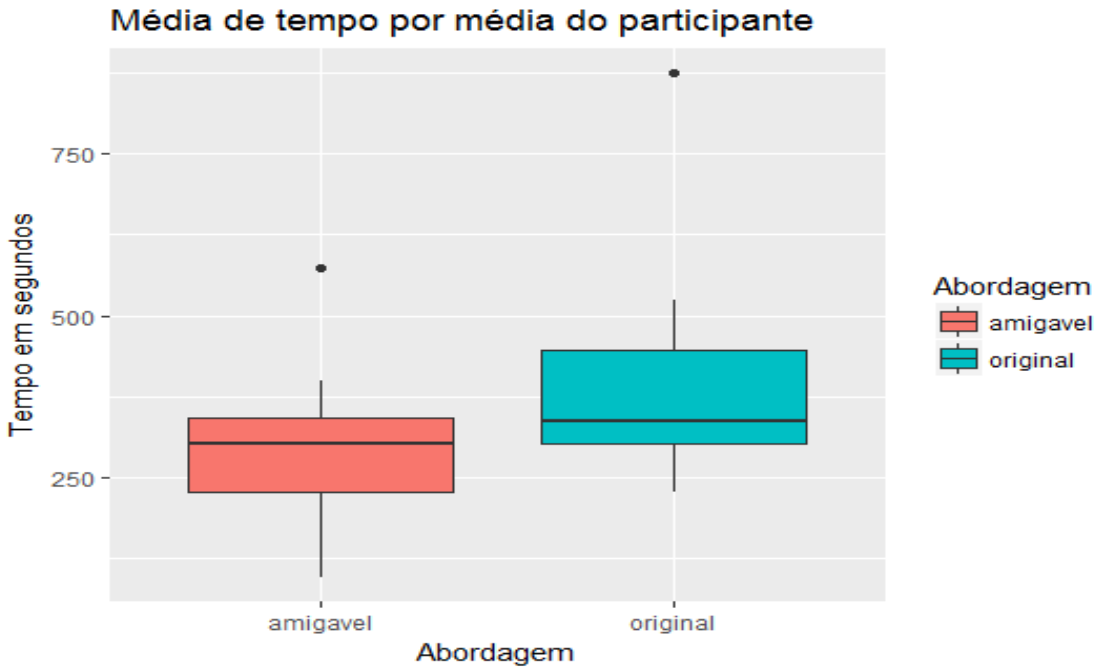

Figura 30 – *Boxplot* da média dos tempos das duas abordagens agrupada pela média obtida pelo participante.

Tabela 10 – Estatística descritiva da média de tempo agrupada pela média do participante.

Abordagem	**Mínimo**	**1^{o} quartil**	**Mediana**	**Média**	**3 ^{o} quartil**	**Máximo**
Amigável	96,0	228,0	301,3	292,6	342,3	573,0
Original	226,8	301,7	337,2	388,8	447,3	874,7

A Figura 31 representa o *boxplot* da quantidade de erros resolvidos das duas abordagens agrupados por participante, de acordo com a Hipótese 4. A figura ilustra uma pequena diferença dos dados, isso ocorreu porque a quantidade de erros resolvidos foi muito próxima, 187 para a abordagem amigável contra 178 da abordagem original. Os dados descritivos das duas abordagens estão ilustrados na Tabela 11.

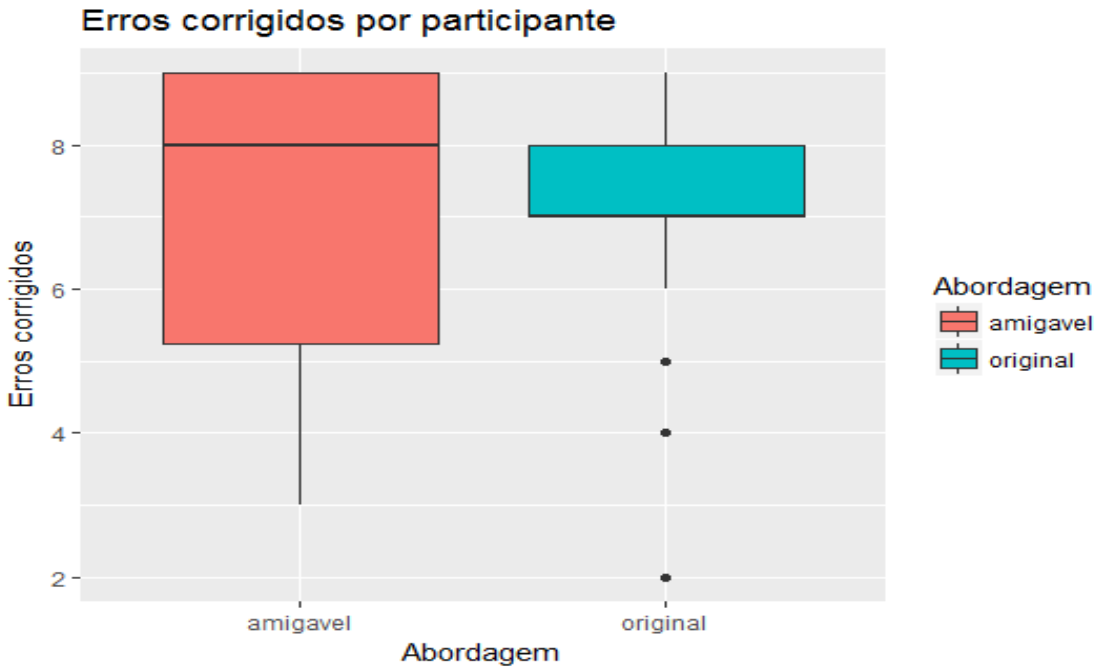

Figura 31 – *Boxplot* da quantidade de erros corrigidos das duas abordagens agrupados por participante.

Tabela 11 – Estatística descritiva dos erros corrigidos agrupados por participante.

Abordagem	**Mínimo**	**1^{o} quartil**	**Mediana**	**Média**	**3 ^{o} quartil**	**Máximo**
Amigável	3,000	5,250	8,000	7,192	9,000	9,000
Original	2,000	7,000	7,000	6,846	8,000	9,000

A Figura 32 representa o *boxplot* da quantidade de erros resolvidos das duas abordagens agrupados pelo nível de inglês dos participantes, de acordo com a Hipótese 5. A figura ilustra pouca diferença entre os dados, apesar do máximo, da média e da mediana da abordagem amigável serem maiores. Os dados estão ilustrados na Tabela 12.

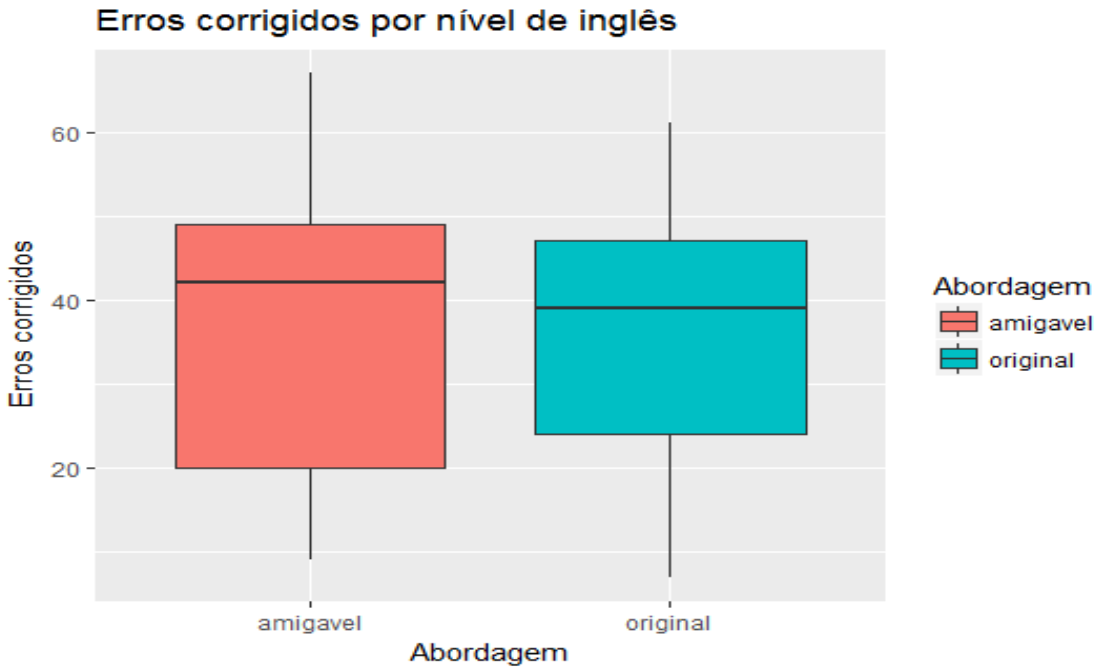

Figura 32 – *Boxplot* da quantidade de erros corrigidos das duas abordagens agrupados por nível de inglês.

Tabela 12 – Estatística descritiva dos erros corrigidos agrupados por nível de inglês.

Abordagem	**Mínimo**	**1^o quartil**	**Mediana**	**Média**	**3 ^{o} quartil**	**Máximo**
Amigável	9,0	20,0	42,0	37,4	49,0	67,0
Original	7,0	24,0	39,0	35,6	47,0	61,0

A Figura 33 representa o *boxplot* da quantidade de erros resolvidos das duas abordagens agrupados pela média dos participantes na disciplina, de acordo com a Hipótese 6. A figura ilustra pouca diferença entre os dados, apesar do mínimo, da mediana, do primeiro quartil, da média e do máximo da abordagem amigável serem um pouco maior, significando que esta abordagem possuiu uma maior quantidade de erros corrigidos, porém sem grande diferença. Os dados estão ilustrados na Tabela 13.

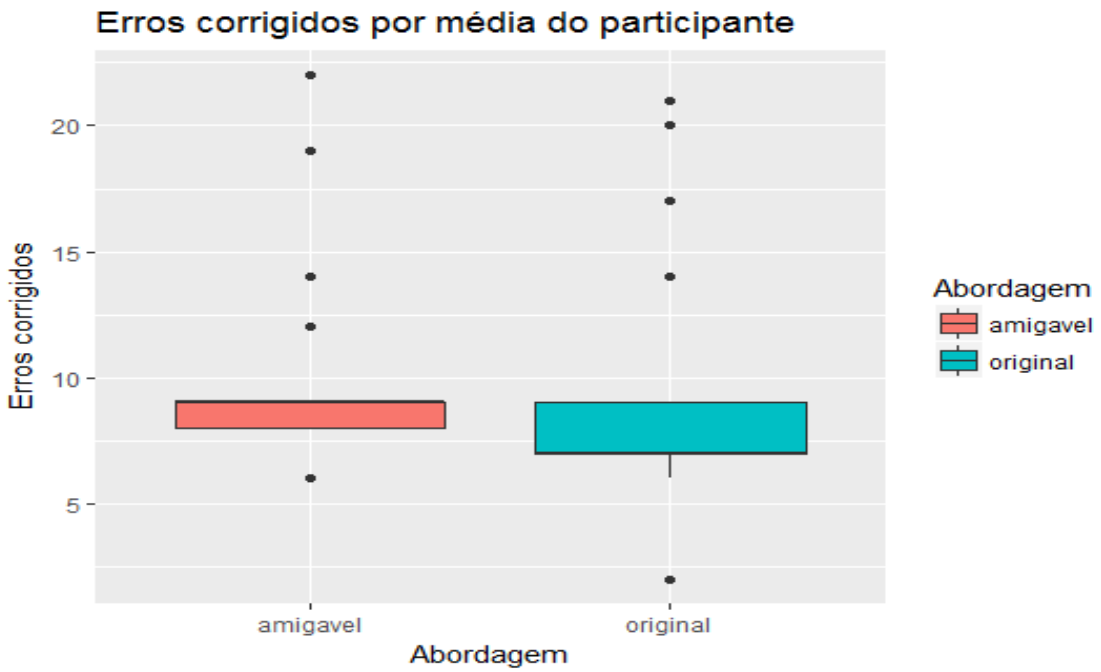

Figura 33 – *Boxplot* da quantidade de erros corrigidos das duas abordagens agrupados pela média do participante.

Tabela 13 – Estatística descritiva dos erros corrigidos agrupados pela média do participante.

Abordagem	**Mínimo**	**1^o quartil**	**Mediana**	**Média**	**3 ^{o} quartil**	**Máximo**
Amigável	6,00	8,00	9,00	10,29	9,00	22,00
Original	2,000	7,000	7,000	9,529	9,000	21,000

5.2 Teste de Normalidade

Antes de realizar os testes de hipóteses da Seção 5.3, foi aplicado o teste de normalidade sob os dados a serem analisados para verificar se seguem uma distribuição normal, a fim de optar pela utilização de testes paramétricos ou não paramétricos em sua avaliação de hipóteses. Os testes paramétricos são utilizados quando os dados geralmente seguem uma distribuição normal, os não paramétricos são utilizados caso contrário. Para verificar a normalidade dos dados é utilizado o teste de normalidade *Shapiro-Wilk* ilustrado no trabalho de (SHAPIRO; WILK, 1965).

Para verificar a normalidade dos dados, são criadas as hipóteses nula e alternativa:

- H_0: A amostra segue uma distribuição normal;
- H_1: A amostra não segue uma distribuição normal;

Com a aplicação do teste de *Shapiro-Wilk*, considerando $\alpha = 0{,}05$, inferindo *p-value* $< \alpha$ para todas as abordagens e agrupamentos analisados diante de suas variáveis de estudo. Sendo assim, a hipótese H_0 é rejeitada caso a sentença *p-value* $< \alpha$ seja verdadeira, caso contrário, não é rejeitada.

Em síntese, se *p-value* $< \alpha$ for verdade, os dados não seguem uma distribuição normal e são recomendados o uso de testes não paramétricos, caso contrário, seguem uma distribuição normal e são recomendados o uso de testes paramétricos.

Diante dos dados da análise de hipóteses, a Tabela 14 ilustra o resultado da aplicação do teste de normalidade dos dados, ilustrando então qual tipo de teste deve ser utilizado de acordo com a inferência encontrada agrupados por participante, nível de inglês e pela média dos participantes, de acordo com a métrica média de tempo em segundos. Já a Tabela 15 aborda a aplicação dos mesmos testes, agrupados com as mesmas características porém com relação à quantidade de erros corrigidos, representando assim, respectivamente o teste de normalidade das hipóteses 1, 2, 3, 4, 5 e 6. A segunda coluna das duas tabelas ilustra o *p-value* de cada agrupamento ao aplicar o teste de *Shapiro-Wilk*. Já a terceira coluna das duas tabelas ilustra a validação de hipótese da normalidade dos dados de acordo com cada agrupamento, negando a hipótese nula quando *p-value* $< \alpha$.

De acordo com o trabalho de Torman, Coster e Riboldi (2012) ao utilizar amostras pareadas, é necessário que a variável aleatória da diferença entre as duas amostras tenha distribuição normal, por isso ao aplicar o teste de *Shapiro-Wilk* para normalidade dos dados, foi utilizada a diferença das duas abordagens.

Como ferramenta para agrupar os dados, foi utilizada a ferramenta Excel, através de recursos como tabelas dinâmicas e agrupamento por filtros. Para aplicação dos testes de normalidade foi utilizada a ferramenta R Studio com a linguagem de programação R, ao usar o comando *shapiro.test(dados)* sob os dados em questão.

Tabela 14 – Aplicação do teste de normalidade *Shapiro-Wilk* com relação ao tempo em segundos.

Agrupamento	**Shapiro(Amigável-Original)**	**Inferência**
Por participante	*p-value* = 0.04839	**Não normal**
Por nível de inglês	*p-value* = 0.5451	**Normal**
Por média	*p-value* = 0.01082	**Não normal**

Tabela 15 – Aplicação do teste de normalidade *Shapiro-Wilk* com relação ao número de erros corrigidos.

Agrupamento	**Shapiro(Amigável-Original)**	**Inferência**
Por participante	*p-value* = 0.1295	**Normal**
Por nível de inglês	*p-value* = 0.4017	**Normal**
Por média	*p-value* = 0.4154	**Normal**

5.3 Validação das Hipóteses

Esta seção apresenta os resultados dos testes das hipóteses levantadas no Capítulo 4 de acordo com a distribuição que os dados seguem.

O Teste de *Wilcoxon Pareado* é um teste não paramétrico para comparação de duas amostras pareadas, que utiliza a diferença entre os valores dos pares de cada amostra para o cálculo da estatística, verificando a magnitude da diferença. A hipótese nula testada é se a mediana da diferença é nula, ou seja, se não há evidências estatísticas de que as amostras diferem. Caso contrário, a hipótese nula é rejeitada e a mediana da diferença não é nula, ou seja, existem evidências de diferenças entre as duas amostras. Este teste é adequado para amostras emparelhadas e quando não é cumprido o pressuposto de normalidade dos dados (WOHLIN et al., 2000).

O *Teste-T Pareado* é um teste paramétrico para comparação de duas amostras pareadas, utilizando os testes de igualdade de variâncias e os testes de médias. Dependendo da hipótese nula, deve ser usado uma ou duas caudas da distribuição *t-Student*. Este teste é adequado para amostras emparelhadas e quando é cumprido o pressuposto de normalidade dos dados (WOHLIN et al., 2000).

Para os dados que não seguem uma distribuição normal, foi utilizado o Teste de *Wilcoxon Pareado* e para os testes que seguem uma distribuição normal, foi utilizado o *Teste-T Pareado*, conforme recomendações de Wohlin et al. (2000), com o objetivo de comparar as duas abordagens de acordo com a variável de agrupamento e a métrica em questão.

Os testes foram aplicados a cada uma das variáveis de análise, a fim de observar estatisticamente a validação das hipóteses com um nível de significância de 5%, o que significa que o resultado é tratado com um grau de confiança de 95%.

Testes com relação ao tempo

Ao aplicar os testes com relação ao tempo, tem-se como objetivo comparar se a média de tempo da abordagem amigável foi **menor** que a original. Para isso, foram utilizados os seguintes parâmetros:

- *amigavel* e *original* representam os dados de entradas com as médias de tempo de acordo com seu respectivo agrupamento;
- *paired=TRUE* representa que está usando o teste com os dados pareados;
- *alternative="**less**"* está aplicando o teste demarcando que a hipótese alternativa seja menor, ou seja, que os dados referentes à abordagem amigável são menores que a original;
- *confLevel=0.95* significa que o nível de confiança é de 95%.

(A) **Hipótese 1:** Dados agrupados por participante
De acordo com as análises apresentadas na Tabela 14, temos que os dados não seguem uma distribuição normal. Por este motivo, foi aplicado o teste de *Wilcoxon Pareado* sob os dados.

Aplicou-se o teste de *Wilcoxon Pareado* na linguagem R: *wilcox.test(amigavel, original, paired=TRUE, alternative="less", confLevel=0.95)*. Foi obtido um ***p-value*** de **0.2262**, sendo **maior** que o α (**0.05**), logo, **não rejeitamos a hipótese H_0**. Com isso, não podemos afirmar que as médias de tempos das duas abordagens com relação aos participantes são diferentes.

(B) **Hipótese 2:** Dados agrupados por nível de inglês

De acordo com as análises apresentadas na Tabela 14, temos que os dados seguem uma distribuição normal. Por este motivo, foi aplicado o teste de *Teste-T Pareado* sob os dados.

Aplicou-se o teste de *Teste-T Pareado* na linguagem R: *t.test(amigavel, original, paired=TRUE, alternative="less", conf.level=0.95)*. Foi obtido um ***p-value*** de **0.01988**, sendo **menor** que o α (**0.05**), logo, **rejeitamos a hipótese H_0**. Com isso, podemos afirmar que as médias de tempos das duas abordagens com relação ao nível de inglês dos participantes são diferentes.

(C) **Hipótese 3:** Dados agrupados por média do aluno

De acordo com as análises apresentadas na Tabela 14, temos que os dados não seguem uma distribuição normal. Por este motivo, foi aplicado o teste de *Wilcoxon Pareado* sob os dados.

Aplicou-se o teste de *Wilcoxon Pareado* na linguagem R: *wilcox.test(amigavel, original, paired=TRUE, alternative="less", confLevel=0.95)*. Foi obtido um ***p-value*** de **0.02325**, sendo **menor** que o α (**0.05**), logo, **rejeitamos a hipótese H_0**. Com isso, podemos afirmar que as médias de tempos das duas abordagens com relação à média de notas dos participantes são diferentes.

A Tabela 16 ilustra uma síntese da aplicação dos testes ilustrando o *p-value* de cada hipótese e a rejeição ou não da hipótese nula.

Testes com relação aos erros corrigidos

Ao aplicar os testes com relação aos erros corrigidos, tem-se como objetivo comparar se a quantidade de erros corrigidos pela abordagem amigável foi **maior** que a original. Para isso, foram utilizados os seguintes parâmetros:

- *amigavel* e *original* representam os dados de entradas com as médias de tempo de acordo com seu respectivo agrupamento;
- *paired=TRUE* representa que está usando o teste com os dados pareados;
- *alternative="**greater**"* está aplicando o teste demarcando que a hipótese alternativa seja maior, ou seja, que os dados referentes à abordagem amigável são maiores que a original;
- *confLevel=0.95* significa que o nível de confiança é de 95%.

(A) **Hipótese 4:** Dados agrupados por participante
De acordo com as análises apresentadas na Tabela 15, temos que os dados seguem uma distribuição normal. Por este motivo, foi aplicado o teste de *Teste-T Pareado* sob os dados.

Aplicou-se o teste de *Teste-T Pareado* na linguagem R: *t.test(amigavel, original, paired=TRUE, alternative="greater", conf.level=0.95)*. Foi obtido um ***p-value*** de **0.2039**, sendo **maior** que o α (**0.05**), logo, **não rejeitamos a hipótese H_0**. Com isso, não podemos afirmar que a quantidade de erros corrigidos das duas abordagens com relação aos participantes são diferentes.

(B) **Hipótese 5:** Dados agrupados por nível de inglês
De acordo com as análises apresentadas na Tabela 15, temos que os dados seguem uma distribuição normal, por este motivo, foi aplicado o teste de *Teste-T Pareado* sob os dados.

Aplicou-se o teste de *Teste-T Pareado* na linguagem R: *t.test(amigavel, original, paired=TRUE, alternative="greater", conf.level=0.95)*. Foi obtido um ***p-value*** de **0.165**, sendo **maior** que o α (**0.05**), logo, **não rejeitamos a hipótese H_0**. Com isso, não podemos afirmar que a quantidade de erros corrigidos das duas abordagens com relação ao nível de inglês dos participantes são diferentes.

(C) **Hipótese 6:** Dados agrupados por média do aluno
De acordo com as análises apresentadas na Tabela 15, temos que os dados seguem uma distribuição normal, por este motivo, foi aplicado o teste de *Teste-T Pareado* sob os dados.

Aplicou-se o teste de *Teste-T Pareado* na linguagem R: *t.test(amigavel, original, paired=TRUE, alternative="greater", conf.level=0.95)*. Foi obtido um ***p-value*** de **0.08503**, sendo **maior** que o α (**0.05**), logo, **não rejeitamos a hipótese H_0**. Com isso, não podemos afirmar que a quantidade de erros corrigidos das duas abordagens com relação à média de notas dos participantes são diferentes.

A Tabela 17 ilustra uma síntese da aplicação dos testes ilustrando o *p-value* de cada hipótese e a rejeição ou não da hipótese nula.

Tabela 16 – Aplicação dos respectivos testes de acordo com a normalização dos dados e à média dos tempos.

Agrupamento	**Teste estatístico**	**Resultado (*p-value*)**	**Inferência**
Por participante	*Wilcoxon Pareado*	**p-value = 0.2262**	**Não rejeitamos a hipótese H_0**
Por nível de inglês	*Teste-T Pareado*	**p-value = 0.01988**	**Rejeitamos a hipótese H_0**
Por média	*Wilcoxon Pareado*	**p-value = 0.02325**	**Rejeitamos a hipótese H_0**

Tabela 17 – Aplicação dos respectivos testes de acordo com a normalização dos dados e à quantidade de erros corrigidos.

Agrupamento	**Teste estatístico**	**Resultado (*p-value*)**	**Inferência**
Por participante	*Teste-T Pareado*	**p-value = 0.2039**	**Não rejeitamos a hipótese H_0**
Por nível de inglês	*Teste-T Pareado*	**p-value = 0.165**	**Não rejeitamos a hipótese H_0**
Por média	*Teste-T Pareado*	**p-value = 0.08503**	**Não rejeitamos a hipótese H_0**

5.4 Ameaças à Validade do Estudo

Ameaças à validade interna

O nível diferenciado de conhecimento dos participantes da linguagem Python pode fazer com que os participantes menos experientes afetem os resultados negativamente. Para sanar este problema, os estudantes que não obtiveram média superior ou igual a cinco na disciplina - apenas 3 (três), foram eliminados da análise de acordo com a média.

O nível de inglês foi definido em questionário aplicado ao final do experimento, porém não foi realizada nenhuma prova para aferir com precisão este valor, ficando a cargo do participante informar seu nível de conhecimento do idioma.

Outra ameaça importante de ilustrar, é com relação ao tempo, pois apesar de estabelecermos os dez minutos máximo por problema, alguns estudantes ultrapassaram este valor, assim como outros desistiram antes do tempo estabelecido, porém isto ocorreu para as duas abordagens.

Não foi contabilizado nos testes experimentais o tempo adicional gasto pelos participantes para clicar no link *ver saída amigável* para acessar a mensagem amigável. É importante ressaltar, que a abordagem original informa a mensagem assim que é feita a submissão. Já na abordagem amigável, é necessário que o participante dê um clique a mais para que a mensagem seja exibida. Esta ação contabiliza cerca de três segundos: clicar no link, aguardar o sistema processar a requisição, exibir a mensagem amigável e finalmente o usuário rolar a tela até o início da mensagem para visualizá-la.

Ameaças à validade externa

O baixo número de participantes representa outra ameaça, visto que pode influenciar negativamente os resultados do experimento. Porém, foi inviável a participação de mais participantes, tanto pela quantidade de laboratórios disponíveis como pelo nível de dificuldade de acompanhar individualmente cada um.

6
Conclusões

Este trabalho de mestrado apresentou trabalhos que abordam o tema de melhora na apresentação de erros sintáticos. Diante dos resultados encontrados a partir do estudo sistemático, foi realizada uma análise na base de dados do juiz *on-line* The Huxley para determinar qual linguagem de programação faria parte da implementação desta funcionalidade. Nesta análise foram encontradas evidências que enfatizaram a escolha da linguagem Python como predileção, visto que foi a linguagem que mais possuiu um percentual de submissões com erros sintáticos na base em questão.

Foram mapeados todos os erros encontrados na base de dados, chegando-se em 144 mensagens de erros, agrupadas em 21 classes. Assim sendo, foi realizada a elucidação dos requisitos funcionais e sua implementação, e também a integração com juiz *on-line* supracitado.

A fim de avaliar as mensagens de erros e dicas elaboradas contra as mensagens originalmente apresentadas pelo juiz *on-line*, foi realizado um estudo experimental. Foi conduzido de acordo com o planejamento estabelecido. Foram utilizadas as mensagens de erro que aparecem com mais frequência na base analisada, sendo utilizado um total de 21 mensagens, representando 87,63% de toda a base. Estas mensagens foram distribuídas em 7 (sete) problemas, sendo que 6 (seis) fizeram parte da análise experimental e 1 (um) como problema exemplo, limitando em 10 (dez) minutos a resolução dos 3 (três) erros existentes na solução de cada problema. A operação do experimento se deu com a participação de 26 (vinte e seis) alunos de turmas reais de programação na última semana de aula disciplina. Também fez parte do estudo experimental a aplicação de um questionário ao final de sua realização, com o objetivo de colher dados qualitativo dos participantes.

Após a finalização do experimento foram obtidos dados numéricos e quantitativos sobre o uso das duas abordagens. De posse destes dados, foram apresentadas a análise descritiva e por fim, a validação das hipóteses propostas por este trabalho, resultando assim, em respostas quantitativas para as questões de pesquisa proposta ao objetivo do experimento:

- Questão 1: Qual é o percentual de soluções que não foram aceitas, de acordo com a base de dados de erros do The Huxley?
- Questão 2: Como o novo *feedback* de mensagens foi recebido pelos alunos?
- Questão 3: As novas mensagens realmente ajudaram na correção dos erros?
- Questão 4: A ferramenta abrange os erros mais comuns da linguagem Python?

Com os resultados da análise da base de dados do The Huxley, foi possível responder a Questão 1, onde cerca 70,30% das submissões realizadas são consideradas como erradas. A Questão 2 foi respondida a partir do questionário aplicado, no geral, todos os estudantes acharam importante que um juiz *on-line* possua mensagens amigáveis, informando que é de extrema importância para uma melhor compreensão do erros, pois mesmo sabendo inglês às vezes a mensagem não é tão clara. A Questão 3 também foi respondida através do questionário, onde cerca de 7,7% dos participantes informaram que sua utilidade foi muito baixa, 23,1% sinalizaram que possui uma utilidade média, cerca de 38,5% informaram que tem uma utilidade alta e por fim, 30,6% que é muito útil. A resposta da Questão 4 foi obtida ao analisar a base de dados que, apesar de abranger todos os erros da base encontrada, não podemos afirmar que abrange os erros mais comuns da linguagem Python, porém abrange praticamente a totalidade dos erros dos usuários que a utilizam na ferramenta de juiz *on-line* já mencionada.

Diante das respostas às questões de pesquisa supracitadas, norteadas pelas hipóteses deste trabalho, indicam que a utilização de mensagens amigáveis pode ajudar no entendimento de mensagens com erros sintáticos, guiando o aluno para sua correção ao apresentar dicas e sugestões de forma mais simples e compreensível. Ao validar as hipóteses de acordo com o nível de inglês e média final obtida na disciplina pelos participantes, foi constatado que as duas abordagens se diferem estatisticamente com relação ao tempo, havendo evidências que uso da abordagem amigável diminui o tempo para solucionar um erro sintático, porém não apresenta diferença significativa estatisticamente quando relacionadas à quantidade de erros resolvidos, assim como não há indícios de diferença estatística dos dados ao comparar os dados por participante com relação ao tempo gasto para resolver um problema.

É importante salientar as respostas obtidas dos participantes com relação a alguns questionamentos. Com relação ao nível de dificuldade em entender as mensagens de erros originais, cerca de 23,1% responderam que é muito alto, 7,7% que é alto, 23,1% que é médio, 36,5% que é baixo e 7,7% que é muito baixo. Já de acordo com o nível de dificuldade em entender as mensagens amigáveis, 3,8% responderam que é muito alto, 0% que é alto, 11,5% que é médio, 26,9% que é baixo e 57,7% que é muito baixo. No tocante ao nível de utilidade do uso das mensagens amigáveis, 30,8% responderam que é muito alto, 38,5% que é alto, 23,1% que é médio, 0% que é baixo e 7,7% que é muito baixo. Desta forma, considerando a opinião

dos participantes do experimento, pode-se concluir que as mensagens amigáveis são mais fáceis de serem entendidas do que as originais e também que foram mais úteis.

Em conformidade com as considerações anteriores, há indícios que no contexto de aprendizes de programação, a utilização da técnica de apresentação de mensagens amigáveis pode diminuir o medo e frustrações, possibilitando assim uma correção mais rápida e melhor guiada. Porém não descartamos o uso das mensagens originais, visto que não foram encontradas diferenças estatísticas evidentes com relação à quantidade de erros corrigidos e ao agrupar por participante com relação ao tempo na realização do estudo experimental.

6.1 Trabalhos Futuros

Como trabalhos futuros, buscando a continuidade da pesquisa e o aperfeiçoamento da proposta deste trabalho destacam-se:

A implementação e replicação do trabalho em outras linguagens de programação aceitas pelo The Huxley, como C, C++, Java, Octave e Pascal, o que já está acontecendo com a linguagem de programação C, visto que já foi realizado o mapeamento na base de dados dos erros nesta linguagem, sendo encontradas cerca de 123 mensagens de erros e agrupadas em 8 (oito) classes de erros (*ExpectedError, TypeError, FunctionError, SyntaxError, GccError, DirectiveError, VariableError e StructureError*), realizando então, a elucidação dos requisitos funcionais. Atualmente estamos em fase de integração e implementação das mensagens de erros encontradas na linguagem C e as desenvolvidas por este trabalho. Posteriormente faremos a integração com o The Huxley e replicação do experimento.

A replicação do experimento que apesar de ter se mostrado consistente a ponto de responder às questões de pesquisa propostas, ainda assim, existem melhorias que podem elevar consideravelmente sua relevância, como por exemplo, o aumento no número de participantes, inserção de uma maior quantidade de erros e problemas, uma vez que os testes de hipótese, quando aplicados em amostras maiores, obtêm resultados mais confiáveis do ponto de vista estatístico.

Referências

AHMADZADEH, M.; ELLIMAN, D.; HIGGINS, C. An analysis of patterns of debugging among novice computer science students. *ACM SIGCSE Bulletin*, ACM, v. 37, n. 3, p. 84–88, 2005. Citado na página 42.

ALEXANDER, K. Rdf in json: a specification for serialising rdf in json. SFSW 2008, 2008. Citado na página 49.

BASILI, V. R.; WEISS, D. M. A methodology for collecting valid software engineering data. *IEEE Transactions on software engineering*, IEEE, n. 6, p. 728–738, 1984. Citado na página 70.

BEZ, J. L.; TONIN, N. A.; RODEGHERI, P. R. Uri online judge academic: A tool for algorithms and programming classes. In: IEEE. *Computer Science & Education (ICCSE), 2014 9th International Conference on*. [S.l.], 2014. p. 149–152. Citado na página 31.

BEZ, J. L.; TONIN, N. A.; RODEGHERI, P. R. URI Online Judge Academic: A Tool for Algorithms and Programming Classes. *The 9th International Conference on Computer Science & Education (ICCSE 2014)*, n. Iccse, p. 149–152, 2014. Citado na página 31.

CAMPOS, C. P. D.; FERREIRA, C. E. Boca: um sistema de apoio a competições de programação. In: *Workshop de Educação em Computação*. [S.l.: s.n.], 2004. p. 885–895. Citado na página 19.

CARDOSO, A. L. M. d. S. Construção e difusão colaborativa do conhecimento: uma experiência construtivista de educação em um ambiente virtual de aprendizagem. 2010. Citado 4 vezes nas páginas 18, 19, 28 e 29.

CHRISTENSEN, E. et al. *Web services description language (WSDL) 1.1*. 2001. Citado na página 61.

GONÇALVES, G. S.; BASTOS, P. H. O.; OLIVEIRA, D. de. O uso de websockets no desenvolvimento de sistemas baseados em uma arquitetura front-end com api. 2014. Citado na página 49.

HRISTOVA, M. et al. Identifying and correcting java programming errors for introductory computer science students. In: ACM. *ACM SIGCSE Bulletin*. [S.l.], 2003. v. 35, n. 1, p. 153–156. Citado 3 vezes nas páginas 9, 40 e 42.

JÚNIOR, J. C. R. P.; RAPKIEWICZ, C. E. O processo de ensino-aprendizagem de fundamentos de programação: Uma visão crítica da pesquisa no brasil. *WEI, I*, v. 7, p. 28, 2004. Citado 2 vezes nas páginas 18 e 20.

KAY, D. G. et al. Automated grading assistance for student programs. *ACM SIGCSE Bulletin*, v. 26, n. 1, p. 381–382, 1994. Citado na página 30.

KJÖLLERSTRÖM, B. Assessment: the key to changing the way we learn. *International Journal of Innovation in Science and Mathematics Education (formerly CAL-laborate International)*, v. 3, n. 1, 2012. Citado na página 31.

KURNIA, A.; LIM, A.; CHEANG, B. Online judge. *Computers & Education*, Elsevier, v. 36, n. 4, p. 299–315, 2001. Citado 2 vezes nas páginas 19 e 30.

LEMOS, M. A. de; BARROS, L. N. de; LOPES, R. de D. Uma biblioteca cognitiva para o aprendizado de programação. 2003. Citado na página 20.

LINO, A. D. et al. Avaliação automática de consultas sql em ambiente virtual de ensinoaprendizagem. In: *Conferencia Ibérica de Sistemas y Tecnologías de la Información. CISTI*. [S.l.: s.n.], 2007. Citado na página 30.

MACHADO, A. C. T. Google docs & spreadsheets: Autoria colaborativa na web 2.0. *e-Tec*, v. 2, n. 1, 2009. Citado na página 35.

MARCEAU, G.; FISLER, K.; KRISHNAMURTHI, S. Measuring the effectiveness of error messages designed for novice programmers. In: ACM. *Proceedings of the 42nd ACM technical symposium on Computer science education*. [S.l.], 2011. p. 499–504. Citado 3 vezes nas páginas 43, 44 e 80.

MARCOLINO, A.; BARBOSA, E. F. Softwares Educacionais para o Ensino de Programação: Um Mapeamento Sistemático. In: *Anais do XXVI Simpósio Brasileiro de Informática na Educação (SBIE 2015)*. [s.n.], 2015. p. 190. Disponível em: <http://br-ie.org/pub/index.php/sbie/article/view/5150>. Citado na página 18.

MIYADERA, Y.; HUANG, N.; YOKOYAMA, S. A programming language education system based on program animation. In: *Proceedings of Educational Uses of Information and Communication Technologie, in IFIP 16th World Computer Congress*. [S.l.: s.n.], 2000. p. 258–261. Citado na página 21.

MOODLE. 2016. Disponível em: <https://docs.moodle.org/31/en/About_Moodle>. Acesso em: 13 nov. 2016. Citado na página 29.

MOREIRA, M. P.; FAVERO, E. L. Um ambiente para ensino de programação com feedback automático de exercícios. In: *Workshop sobre Educação em Computação (WEI 2009)*. [S.l.: s.n.], 2009. v. 17. Citado 3 vezes nas páginas 19, 30 e 31.

ODEKIRK-HASH, E.; ZACHARY, J. L. Automated feedback on programs means students need less help from teachers. In: ACM. *ACM SIGCSE Bulletin*. [S.l.], 2001. v. 33, n. 1, p. 55–59. Citado na página 38.

PAES, R. d. B. et al. Ferramenta para a Avaliação de Aprendizado de Alunos em Programação de Computadores. In: *Anais dos Workshops do Congresso Brasileiro de Informática na Educação (WCBIE 2013)*. [s.n.], 2013. p. 203–212. Disponível em: <http://www.br-ie.org/pub/index.php/wcbie/article/view/2669>. Citado 4 vezes nas páginas 9, 31, 32 e 33.

PELZ, F. D. *Um gerador de dicas para guiar novatos na aprendizagem de programação*. Tese (Dissertação (Mestrado em Computação Aplicada)) — Universidade do Vale do Itajaí, 2014. Citado na página 44.

PENTERICH, E. Ambientes virtuais de aprendizagem. *Sala de Aula e Tecnologias. São Paulo: Editora da Universidade Metodista de São Paulo*, 2005. Citado na página 29.

PEREIRA, A. T. C.; SCHMITT, V.; DIAS, M. Ambientes virtuais de aprendizagem. *AVA-Ambientes Virtuais de Aprendizagem em Diferentes Contextos. Rio de Janeiro: Editora Ciência Moderna Ltda*, p. 23, 2007. Citado 4 vezes nas páginas 9, 19, 28 e 29.

PRESSMAN, R. S. *Engenharia de software - Uma Abordagem Profissional*. [S.l.]: Amgh Editora, 2016. v. 8. Citado na página 53.

PRIOR, J. C. Online assessment of sql query formulation skills. In: AUSTRALIAN COMPUTER SOCIETY, INC. *Proceedings of the fifth Australasian conference on Computing education-Volume 20*. [S.l.], 2003. p. 247–256. Citado 2 vezes nas páginas 20 e 30.

RIBEIRO, E. N.; MENDONÇA, G. d. A.; MENDONÇA, A. F. A importância dos ambientes virtuais de aprendizagem na busca de novos domínios da ead. In: *Anais do 13º Congresso Internacional de Educação a Distância. Curitiba, Brasil*. [S.l.: s.n.], 2007. Citado na página 29.

SANNER, M. F. et al. Python: a programming language for software integration and development. *J Mol Graph Model*, v. 17, n. 1, p. 57–61, 1999. Citado na página 34.

SANTOS, J. C.; RIBEIRO, A. R. Jonline: proposta preliminar de um juiz online didático para o ensino de programação. *Simpósio Brasileiro de Informática na Educação (SBIE 2011)*, v. 22, p. 48, 2011. Citado 2 vezes nas páginas 19 e 30.

SCHORSCH, T. Cap: an automated self-assessment tool to check pascal programs for syntax, logic and style errors. In: ACM. *ACM SIGCSE Bulletin*. [S.l.], 1995. v. 27, n. 1, p. 168–172. Citado 4 vezes nas páginas 9, 36, 37 e 38.

SHAPIRO, S. S.; WILK, M. B. An analysis of variance test for normality (complete samples). *Biometrika*, JSTOR, v. 52, n. 3/4, p. 591–611, 1965. Citado na página 89.

SILVA, C. P. d. C. BOCA Ulisses Furquim Freire da. Boca online contest administrator. In: . [s.n.], 2016. Disponível em: <maratona.ime.usp.br/manualBOCA.html>. Citado 3 vezes nas páginas 12, 30 e 31.

STUDIO, P. *Portugol Studio*. 2017. Disponível em: <http://lite.acad.univali.br/portugol/>. Acesso em: 10 fev. 2017. Citado na página 44.

STUDIO, R. *R Studio - Open source and enterprise-ready professional software for R*. 2017. Disponível em: <https://www.rstudio.com/>. Acesso em: 17 fev. 2017. Citado na página 35.

TONIN, N. A.; BEZ, J. L. Uri online judge: A new interactive learning approach. *Computer Technology and Application*, David Publishing Company, Inc., v. 4, n. 1, 2013. Citado na página 31.

TORMAN, V. B. L.; COSTER, R.; RIBOLDI, J. Normalidade de variáveis: métodos de verificação e comparação de alguns testes não-paramétricos por simulação. *Clinical & Biomedical Research*, v. 32, n. 2, 2012. Citado na página 90.

VINOSKI, S. Chain of responsibility. *IEEE Internet Computing*, IEEE, v. 6, n. 6, p. 80–83, 2002. Citado na página 59.

WEBER, G.; BRUSILOVSKY, M.; STEINLE, F. Elm-pe: An intelligent learning environment for programming, 1996. *Obtain in: www. psychologie. uni-trier. de*, v. 8000, 2014. Citado na página 20.

WOHLIN, C. et al. *Experimentation in software engineering: an introduction. 2000*. [S.l.]: Kluwer Academic Publishers, 2000. Citado 5 vezes nas páginas 69, 70, 71, 74 e 92.

YUJIAN, L.; BO, L. A normalized levenshtein distance metric. *IEEE transactions on pattern analysis and machine intelligence*, IEEE, v. 29, n. 6, p. 1091–1095, 2007. Citado na página 44.

Apêndices

APÊNDICE A – Códigos-fontes do Experimento

A seguir são ilustrados os códigos-fontes disponibilizados aos participantes do experimento contendo cada erro sintático, assim como os códigos-fontes correspondentes após a correção de seus erros.

Problema 1: Elevador Seguro

A Listagem A.1 ilustra o código-fonte para o problema 1, onde são apresentados 3 (três) erros sintáticos, são eles:

- #erro 1, linha 7 - TypeError: unorderable types: str() > int();
- #erro 2, linha 10 - SyntaxError: invalid syntax;
- #erro 3, linha 12 - IndentationError: unexpected indent.

```
1  pesopp = float(input())
2  if pesopp>0:
3      qtp=1
4  elif pesopp<=0:
5      qtp=0
6  while ((pesopp<560) and (qtp<7 or qtp==0)):
7      p = input() #erro 1
8      if p == 0:
9          break
10     elif p>0 #erro 2
11         pesopp = pesopp + p
12           qtp = qtp +1 # erro 3
13 print(qtp)
14 print(pesopp)
```

Listagem A.1 – Código-fonte para o problema 1 (um), apresentando 3 (três) erros sintáticos.

A Listagem A.2 ilustra o código-fonte após a correção dos 3 (três) erros supracitados.

```
1  pesopp = float(input())
2  if pesopp>0:
3      qtp=1
4  elif pesopp<=0:
```

```
    qtp=0
while ((pesopp<560) and (qtp<7 or qtp==0)):
    p = float(input())
    if p == 0:
        break
    elif p>0:
        pesopp = pesopp + p
        qtp = qtp +1
print(qtp)
print(pesopp)
```

Listagem A.2 – Código-fonte para o problema 1 (um) com os erros corrigidos.

Problema 2: Aprovados no Concurso

A Listagem A.3 ilustra o código-fonte para o problema 2, onde são apresentados 3 (três) erros sintáticos, são eles:

- #erro 1, linha 2 - NameError: name 'P' is not defined;
- #erro 2, linha 5 - ValueError: invalid literal for int() with base 10: '8.3';
- #erro 3, linha 9 - SyntaxError: Missing parentheses in call to 'print'.

```
AP=0
p = int(input()) #erro 1
while P>0:
    M = int(input())
    R = int(input()) #erro 2
    if P >= 40 and M >= 21 and R >= 7:
        AP+=1
    P = int(input())
print AP #erro 3
```

Listagem A.3 – Código-fonte para o problema 2 (dois), apresentando 3 (três) erros sintáticos.

A Listagem A.4 ilustra o código-fonte após a correção dos 3 (três) erros supracitados.

```
AP=0
P = int(input())
while P>0:
    M = int(input())
    R = float(input())
    if P >= 40 and M >= 21 and R >= 7:
        AP+=1
    P = int(input())
```

```
9 print(AP)
```

Listagem A.4 – Código-fonte para o problema 2 (dois) com os erros corrigidos.

Problema 3: Caracteres

A Listagem A.5 ilustra o código-fonte para o problema 3, onde são apresentados 3 (três) erros sintáticos, são eles:

- #erro 1, linha 3 - IndentationError: expected an indented block;
- #erro 2, linha 4 - IndexError: string index out of range;
- #erro 3, linha 7 - TypeError: unsupported operand type(s) for -: 'int' and 'str'.

```
1 n = int (input())
2 while n!= 0:
3 letras = input() #erro 1
4     i = len(letras) #erro 2
5     while(i >= 0):
6         print(letras[i], end='')
7         i = i -  '1' #erro 3
8     print()
9     n = int (input())
```

Listagem A.5 – Código-fonte para o problema 3 (três), apresentando 3 (três) erros sintáticos.

A Listagem A.6 ilustra o código-fonte após a correção dos 3 (três) erros supracitados.

```
1 n = int (input())
2 while n!= 0:
3     letras = input()
4     i = len(letras)-1
5     while(i >= 0):
6         print(letras[i], end='')
7         i = i - 1
8     print()
9     n = int (input())
```

Listagem A.6 – Código-fonte para o problema 3 (três) com os erros corrigidos.

Problema 4: Desafio do menor e maior número

A Listagem A.7 ilustra o código-fonte para o problema 4, onde são apresentados 3 (três) erros sintáticos, são eles:

- #erro 1, linha 1 - AttributeError: 'str' object has no attribute 'splite';

- #erro 2, linha 5 - TypeError: 'int' object is not iterable;
- #erro 3, linha 10 - IndentationError: unindent does not match any outer indentation level.

```
1  linha = input().splite() #erro 1
2  lista = [int(valor) for valor in linha]
3  menor = 99999
4  maior = -99999
5  for i in (len(lista)-1): #erro 2
6      if (lista[i] > maior):
7          maior = lista[i]
8      if (lista[i] < menor):
9          menor = lista[i]
10   print(menor, maior) #erro 3
```

Listagem A.7 – Código-fonte para o problema 4 (quatro), apresentando 3 (três) erros sintáticos.

A Listagem A.8 ilustra o código-fonte após a correção dos 3 (três) erros supracitados.

```
1  linha = input().split()
2  lista = [int(valor) for valor in linha]
3  menor = 99999
4  maior = -99999
5  for i in range(len(lista)-1):
6      if (lista[i] > maior):
7          maior = lista[i]
8      if (lista[i] < menor):
9          menor = lista[i]
10 print(menor, maior)
```

Listagem A.8 – Código-fonte para o problema 4 (quatro) com os erros corrigidos.

Problema 5: Equação do Segundo Grau

A Listagem A.9 ilustra o código-fonte para o problema 5, onde são apresentados 3 (três) erros sintáticos, são eles:

- #erro 1, nas linhas 5 e 6 - ZeroDivisionError: float division by zero;
- #erro 2, nas linhas 11 e 12 - TypeError: not all arguments converted during string formatting;
- #erro 3, linha 14 - SyntaxError: unexpected EOF while parsing.

```
1  a=float(input())
2  b=float(input())
3  c=float(input())
4  d=(b**2)-(4*a*c)
5  x1=((-b)+d**(1/2))/(2*a) #erro 1
6  x2=((-b)-d**(1/2))/(2*a) #erro 1
7  if a==0:
8      print('NEESG')
9  else:
10     if d>=0:
11         print("%.2f"%x1) #erro 2
12         print("%.2f"%x2) #erro 2
13     else:
14         print(('NRR') #erro 3
```

Listagem A.9 – Código-fonte para o problema 5 (cinco), apresentando 3 (três) erros sintáticos.

A Listagem A.10 ilustra o código-fonte após a correção dos 3 (três) erros supracitados.

```
1  =float(input())
2  b=float(input())
3  c=float(input())
4  d=(b**2)-(4*a*c)
5  if a==0:
6      print('NEESG')
7  else:
8      if d>=0:
9          x1=((-b)+d**(1/2))/(2*a)
10         x2=((-b)-d**(1/2))/(2*a)
11         print("%.2f"%x1)
12         print("%.2f"%x2)
13     else:
14         print('NRR')
```

Listagem A.10 – Código-fonte para o problema 5 (cinco) com os erros corrigidos.

Problema 6: Balanceando números

A Listagem A.11 ilustra o código-fonte para o problema 6, onde são apresentados 3 (três) erros sintáticos, são eles:

- #erro 1, linha 1 - ValueError: could not convert string to float: '1 2 2';
- #erro 2, linha 7 - EOFError: EOF when reading a line;
- #erro 3, linha 11 - KeyError: 7.

```
linha1 = float(input()) #erro 1
a, f, t = int(linha1[0]), int(linha1[1]), int(linha1[2])
num = input().split()
numbers = {}
cont = 0
for x in num:
    numbers[cont] = (int(input(x))) #erro 2
    cont = cont + 1
conta = 0
ok = True
for pos in range(1, cont+1): #erro 3
    if int(numbers[pos]) == a:
        conta += 1
    else:
        conta -= 1
    if conta < 0:
        ok = False
        break
if conta == 0 and ok:
    print('Balanceada!')
else:
    print('Desbalanceada!')
```

Listagem A.11 – Código-fonte para o problema 6 (seis), apresentando 3 (três) erros sintáticos.

A Listagem A.12 ilustra o código-fonte após a correção dos 3 (três) erros supracitados.

```
linha1 = input().split()
a, f, t = int(linha1[0]), int(linha1[1]), int(linha1[2])
num = input().split()
numbers = {}
cont = 0
for x in num:
    numbers[cont] = (int(x))
    cont = cont + 1
conta = 0
ok = True
for pos in range(0, cont):
    if int(numbers[pos]) == a:
        conta += 1
    else:
        conta -= 1
    if conta < 0:
        ok = False
```

```
18          break
19 if conta == 0 and ok:
20     print('Balanceada!')
21 else:
22     print('Desbalanceada!')
```

Listagem A.12 – Código-fonte para o problema 6 (seis) com os erros corrigidos.

APÊNDICE B – Execução do experimento

A seguir são ilustradas imagens relativas à execução do experimento, ilustrando as mensagens exibidas pelo juiz *on-line* The Huxley.

Problema Exemplo: Mínimo múltiplo comum

As Figuras 34, 36 e 38 ilustram a exibição das mensagens de erros da abordagem original referentes aos erros do problema exemplo, apresentados durante a execução do experimento. Já as Figuras 35, 37 e 39 ilustram a exibição das mensagens de erros da abordagem amigável referentes aos erros do mesmo problema, apresentados durante a execução do experimento, ao clicar no link "*ver saída amigável*", localizado no canto inferior esquerdo das figuras.

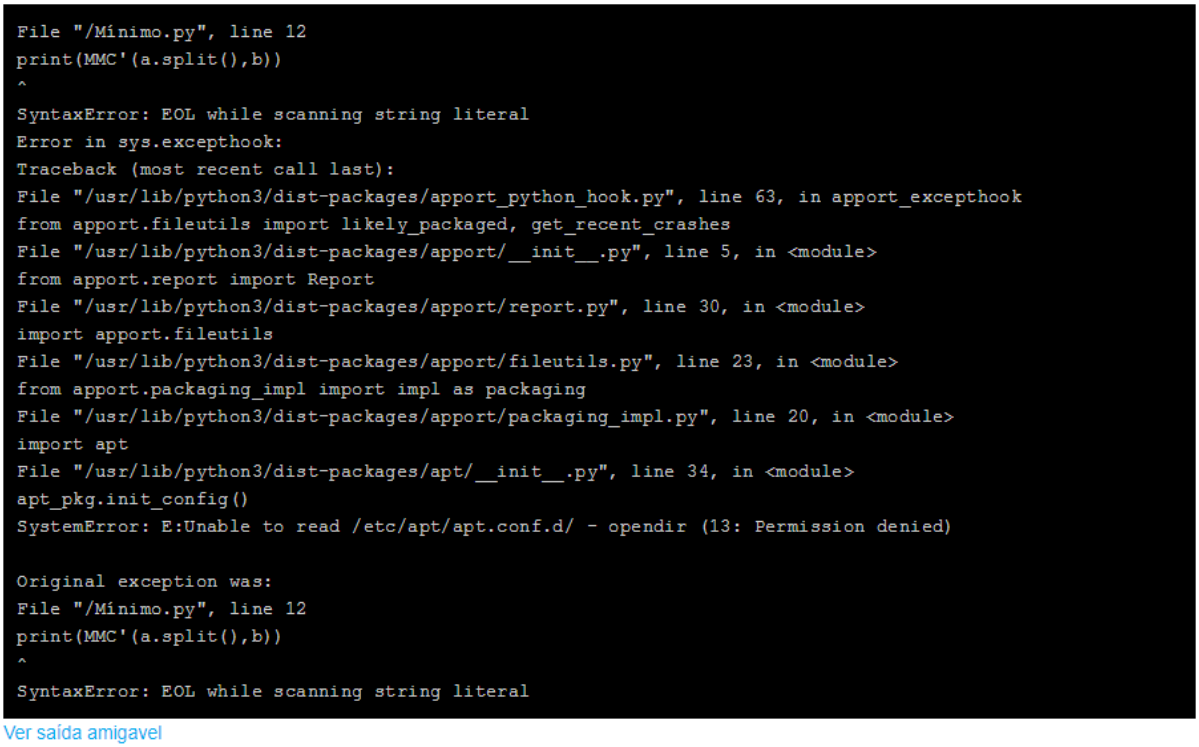

Figura 34 – Exibição de erro sintático da abordagem original no problema exemplo.

```
Erro na linha 12
No trecho de código:
print(MMC'(a.split(),b))
^

Descrição: EOL - End of line - o final de linha da declaração não foi fechado corretamente, insira "
(aspas) em seu código para solucionar o problema e verifque se o número de aspas é par.

Erro na linha 34
No trecho de código:
apt_pkg.init_config()

Descrição: Houve erro de permissão do sistema para leitura do arquivo no caminho E:Unable to read
/etc/apt/apt.conf.d/ - ao abrir o diretório e executar os testes no servidor.

Erro na linha 12
No trecho de código:
print(MMC'(a.split(),b))
^

Descrição: EOL - End of line - o final de linha da declaração não foi fechado corretamente, insira "
(aspas) em seu código para solucionar o problema e verifque se o número de aspas é par.
```

Ver saída original

Figura 35 – Exibição de erro sintático da abordagem amigável no problema exemplo.

```
Traceback (most recent call last):
File "/Mínimo.py", line 10, in <module>
a = int(nums[0].split(int()))
TypeError: Can't convert 'int' object to str implicitly
Error in sys.excepthook:
Traceback (most recent call last):
File "/usr/lib/python3/dist-packages/apport_python_hook.py", line 63, in apport_excepthook
from apport.fileutils import likely_packaged, get_recent_crashes
File "/usr/lib/python3/dist-packages/apport/__init__.py", line 5, in <module>
from apport.report import Report
File "/usr/lib/python3/dist-packages/apport/report.py", line 30, in <module>
import apport.fileutils
File "/usr/lib/python3/dist-packages/apport/fileutils.py", line 23, in <module>
from apport.packaging_impl import impl as packaging
File "/usr/lib/python3/dist-packages/apport/packaging_impl.py", line 20, in <module>
import apt
File "/usr/lib/python3/dist-packages/apt/__init__.py", line 34, in <module>
apt_pkg.init_config()
SystemError: E:Unable to read /etc/apt/apt.conf.d/ - opendir (13: Permission denied)

Original exception was:
Traceback (most recent call last):
File "/Mínimo.py", line 10, in <module>
a = int(nums[0].split(int()))
TypeError: Can't convert 'int' object to str implicitly
```

Ver saída amigavel

Figura 36 – Exibição de erro sintático da abordagem original no problema exemplo.

```
Erro na linha 10
No trecho de código:
a = int(nums[0].split(int()))

Descrição: Não é possível converter o tipo int implicitamente para o tipo str, ou seja, por mais que
seja compreensível sua conversão.

Erro na linha 34
No trecho de código:
apt_pkg.init_config()

Descrição: Houve erro de permissão do sistema para leitura do arquivo no caminho E:Unable to read
/etc/apt/apt.conf.d/ - ao abrir o diretório e executar os testes no servidor.

Erro na linha 10
No trecho de código:
a = int(nums[0].split(int()))

Descrição: Não é possível converter o tipo int implicitamente para o tipo str, ou seja, por mais que
seja compreensível sua conversão.
```

Ver saída original

Figura 37 – Exibição de erro sintático da abordagem amigável no problema exemplo.

```
Traceback (most recent call last):
File "/Mínimo.py", line 10, in <module>
a = int(nums[0].split())
TypeError: int() argument must be a string, a bytes-like object or a number, not 'list'
Error in sys.excepthook:
Traceback (most recent call last):
File "/usr/lib/python3/dist-packages/apport_python_hook.py", line 63, in apport_excepthook
from apport.fileutils import likely_packaged, get_recent_crashes
File "/usr/lib/python3/dist-packages/apport/__init__.py", line 5, in <module>
from apport.report import Report
File "/usr/lib/python3/dist-packages/apport/report.py", line 30, in <module>
import apport.fileutils
File "/usr/lib/python3/dist-packages/apport/fileutils.py", line 23, in <module>
from apport.packaging_impl import impl as packaging
File "/usr/lib/python3/dist-packages/apport/packaging_impl.py", line 20, in <module>
import apt
File "/usr/lib/python3/dist-packages/apt/__init__.py", line 34, in <module>
apt_pkg.init_config()
SystemError: E:Unable to read /etc/apt/apt.conf.d/ - opendir (13: Permission denied)

Original exception was:
Traceback (most recent call last):
File "/Mínimo.py", line 10, in <module>
a = int(nums[0].split())
TypeError: int() argument must be a string, a bytes-like object or a number, not 'list'
```

Ver saída amigavel

Figura 38 – Exibição de erro sintático da abordagem original no problema exemplo.

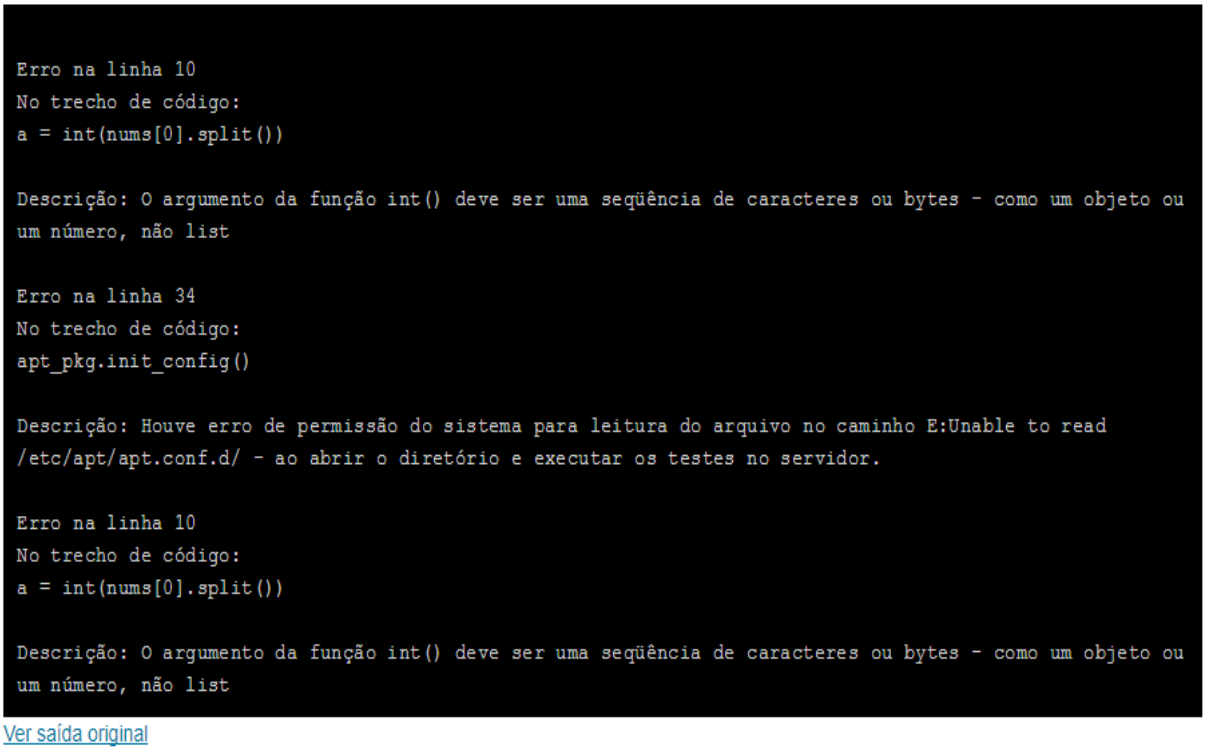

Figura 39 – Exibição de erro sintático da abordagem amigável no problema exemplo.

Problema 1: Elevador Seguro

As Figuras 40, 42 e 44 ilustram a exibição das mensagens de erros da abordagem original referentes aos erros do problema 1 (um), apresentados durante a execução do experimento. Já as Figuras 41, 43 e 45 ilustram a exibição das mensagens de erros da abordagem amigável referentes aos erros do mesmo problema, apresentados durante a execução do experimento, ao clicar no link "*ver saída amigável*", localizado no canto inferior esquerdo das figuras.

```
File "/Elevador.py", line 10
elif p>0
^
SyntaxError: invalid syntax
Error in sys.excepthook:
Traceback (most recent call last):
File "/usr/lib/python3/dist-packages/apport_python_hook.py", line 63, in apport_excepthook
from apport.fileutils import likely_packaged, get_recent_crashes
File "/usr/lib/python3/dist-packages/apport/__init__.py", line 5, in <module>
from apport.report import Report
File "/usr/lib/python3/dist-packages/apport/report.py", line 30, in <module>
import apport.fileutils
File "/usr/lib/python3/dist-packages/apport/fileutils.py", line 23, in <module>
from apport.packaging_impl import impl as packaging
File "/usr/lib/python3/dist-packages/apport/packaging_impl.py", line 20, in <module>
import apt
File "/usr/lib/python3/dist-packages/apt/__init__.py", line 34, in <module>
apt_pkg.init_config()
SystemError: E:Unable to read /etc/apt/apt.conf.d/ - opendir (13: Permission denied)

Original exception was:
File "/Elevador.py", line 10
elif p>0
^
SyntaxError: invalid syntax
```

Ver saída amigavel

Figura 40 – Exibição de erro sintático da abordagem original no problema 1.

```
Erro na linha 10
No trecho de código:
elif p>0
^

Descrição: Verifique se está faltando algum parênteses, a ausência de "":"" e declaração correta de
alguns comandos, tais como: atribuição, if, for, while, input, print e outros.

Erro na linha 34
No trecho de código:
apt_pkg.init_config()

Descrição: Houve erro de permissão do sistema para leitura do arquivo no caminho E:Unable to read
/etc/apt/apt.conf.d/ - ao abrir o diretório e executar os testes no servidor.

Erro na linha 10
No trecho de código:
elif p>0
^

Descrição: Verifique se está faltando algum parênteses, a ausência de "":"" e declaração correta de
alguns comandos, tais como: atribuição, if, for, while, input, print e outros.
```

Ver saída original

Figura 41 – Exibição de erro sintático da abordagem amigável no problema 1.

```
File "/Elevador.py", line 12
qtp = qtp +1
^
IndentationError: unexpected indent
Error in sys.excepthook:
Traceback (most recent call last):
File "/usr/lib/python3/dist-packages/apport_python_hook.py", line 63, in apport_excepthook
from apport.fileutils import likely_packaged, get_recent_crashes
File "/usr/lib/python3/dist-packages/apport/__init__.py", line 5, in <module>
from apport.report import Report
File "/usr/lib/python3/dist-packages/apport/report.py", line 30, in <module>
import apport.fileutils
File "/usr/lib/python3/dist-packages/apport/fileutils.py", line 23, in <module>
from apport.packaging_impl import impl as packaging
File "/usr/lib/python3/dist-packages/apport/packaging_impl.py", line 20, in <module>
import apt
File "/usr/lib/python3/dist-packages/apt/__init__.py", line 34, in <module>
apt_pkg.init_config()
SystemError: E:Unable to read /etc/apt/apt.conf.d/ - opendir (13: Permission denied)

Original exception was:
File "/Elevador.py", line 12
qtp = qtp +1
^
IndentationError: unexpected indent
```

Ver saída amigavel

Figura 42 – Exibição de erro sintático da abordagem original no problema 1.

```
Erro na linha 12
No trecho de código:
qtp = qtp +1
^

Descrição: Indentação inesperada, por favor verifique a indentação. A indentação é uma característica
peculiar na linguagem. Enquanto que os blocos são delimitados explicitamente em C, Java e PHP por chaves
e em Pascal e Fortran por palavras-chave como then e endif, em Python blocos são delimitados por espaços
ou tabulações formando uma indentação visual. Não existem símbolos de "abre" e "fecha". Fique atento
para não errar na próxima!

Erro na linha 34
No trecho de código:
apt_pkg.init_config()

Descrição: Houve erro de permissão do sistema para leitura do arquivo no caminho E:Unable to read
/etc/apt/apt.conf.d/ - ao abrir o diretório e executar os testes no servidor.

Erro na linha 12
No trecho de código:
qtp = qtp +1
^

Descrição: Indentação inesperada, por favor verifique a indentação. A indentação é uma característica
peculiar na linguagem. Enquanto que os blocos são delimitados explicitamente em C, Java e PHP por chaves
e em Pascal e Fortran por palavras-chave como then e endif, em Python blocos são delimitados por espaços
ou tabulações formando uma indentação visual. Não existem símbolos de "abre" e "fecha". Fique atento
para não errar na próxima!
```

Ver saída original

Figura 43 – Exibição de erro sintático da abordagem amigável no problema 1.

```
Traceback (most recent call last):
File "/Elevador.py", line 10, in <module>
elif p>0:
TypeError: unorderable types: str() > int()
Error in sys.excepthook:
Traceback (most recent call last):
File "/usr/lib/python3/dist-packages/apport_python_hook.py", line 63, in apport_excepthook
from apport.fileutils import likely_packaged, get_recent_crashes
File "/usr/lib/python3/dist-packages/apport/__init__.py", line 5, in <module>
from apport.report import Report
File "/usr/lib/python3/dist-packages/apport/report.py", line 30, in <module>
import apport.fileutils
File "/usr/lib/python3/dist-packages/apport/fileutils.py", line 23, in <module>
from apport.packaging_impl import impl as packaging
File "/usr/lib/python3/dist-packages/apport/packaging_impl.py", line 20, in <module>
import apt
File "/usr/lib/python3/dist-packages/apt/__init__.py", line 34, in <module>
apt_pkg.init_config()
SystemError: E:Unable to read /etc/apt/apt.conf.d/ - opendir (13: Permission denied)

Original exception was:
Traceback (most recent call last):
File "/Elevador.py", line 10, in <module>
elif p>0:
TypeError: unorderable types: str() > int()
```

Ver saída amigavel

Figura 44 – Exibição de erro sintático da abordagem original no problema 1.

```
Erro na linha 10
No trecho de código:
elif p>0:

Descrição: Os tipos não são comparáveis, por isso não é possível realizar a comparação: str() > int().
Modifique-os para que sejam do mesmo tipo.

Erro na linha 34
No trecho de código:
apt_pkg.init_config()

Descrição: Houve erro de permissão do sistema para leitura do arquivo no caminho E:Unable to read
/etc/apt/apt.conf.d/ - ao abrir o diretório e executar os testes no servidor.

Erro na linha 10
No trecho de código:
elif p>0:

Descrição: Os tipos não são comparáveis, por isso não é possível realizar a comparação: str() > int().
Modifique-os para que sejam do mesmo tipo.
```

Ver saída original

Figura 45 – Exibição de erro sintático da abordagem amigável no problema 1.

Problema 2: Aprovados no Concurso

As Figuras 46, 48 e 50 ilustram a exibição das mensagens de erros da abordagem original referentes aos erros do problema 2 (dois), apresentados durante a execução do experimento. Já as Figuras 47, 49 e 51 ilustram a exibição das mensagens de erros da abordagem amigável referentes aos erros do mesmo problema, apresentados durante a execução do experimento, ao clicar no link "*ver saída amigável*", localizado no canto inferior esquerdo das figuras.

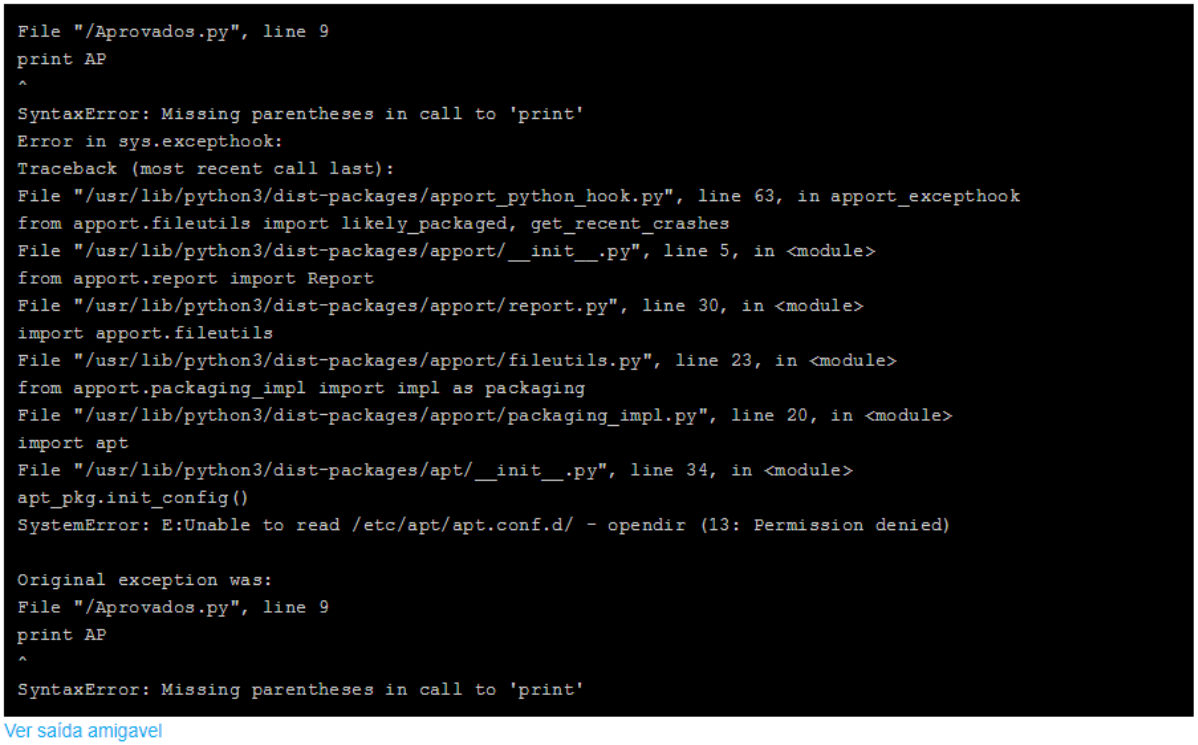

Figura 46 – Exibição de erro sintático da abordagem original no problema 2.

```
Erro na linha 9
No trecho de código:
print AP
^

Descrição: Verifique o uso do parêntese, a partir da versão 3.0 do Python seu uso é obrigatório no
comando print.

Erro na linha 34
No trecho de código:
apt_pkg.init_config()

Descrição: Houve erro de permissão do sistema para leitura do arquivo no caminho E:Unable to read
/etc/apt/apt.conf.d/ - ao abrir o diretório e executar os testes no servidor.

Erro na linha 9
No trecho de código:
print AP
^

Descrição: Verifique o uso do parêntese, a partir da versão 3.0 do Python seu uso é obrigatório no
comando print.
```

Ver saída original

Figura 47 – Exibição de erro sintático da abordagem amigável no problema 2.

```
Traceback (most recent call last):
File "/Aprovados.py", line 3, in <module>
while P>0:
NameError: name 'P' is not defined
Error in sys.excepthook:
Traceback (most recent call last):
File "/usr/lib/python3/dist-packages/apport_python_hook.py", line 63, in apport_excepthook
from apport.fileutils import likely_packaged, get_recent_crashes
File "/usr/lib/python3/dist-packages/apport/__init__.py", line 5, in <module>
from apport.report import Report
File "/usr/lib/python3/dist-packages/apport/report.py", line 30, in <module>
import apport.fileutils
File "/usr/lib/python3/dist-packages/apport/fileutils.py", line 23, in <module>
from apport.packaging_impl import impl as packaging
File "/usr/lib/python3/dist-packages/apport/packaging_impl.py", line 20, in <module>
import apt
File "/usr/lib/python3/dist-packages/apt/__init__.py", line 34, in <module>
apt_pkg.init_config()
SystemError: E:Unable to read /etc/apt/apt.conf.d/ - opendir (13: Permission denied)

Original exception was:
Traceback (most recent call last):
File "/Aprovados.py", line 3, in <module>
while P>0:
NameError: name 'P' is not defined
```

Ver saída amigavel

Figura 48 – Exibição de erro sintático da abordagem original no problema 2.

```
Erro na linha 3
No trecho de código:
while P>0:

Descrição: Foi feito o uso de uma variável que não foi definida ou um comando que foi escrito de forma
errada. Verifique a palavra 'P'.

Erro na linha 34
No trecho de código:
apt_pkg.init_config()

Descrição: Houve erro de permissão do sistema para leitura do arquivo no caminho E:Unable to read
/etc/apt/apt.conf.d/ - ao abrir o diretório e executar os testes no servidor.

Erro na linha 3
No trecho de código:
while P>0:

Descrição: Foi feito o uso de uma variável que não foi definida ou um comando que foi escrito de forma
errada. Verifique a palavra 'P'.
```

Ver saída original

Figura 49 – Exibição de erro sintático da abordagem amigável no problema 2.

```
Traceback (most recent call last):
File "/Aprovados.py", line 5, in <module>
R = int(input())
ValueError: invalid literal for int() with base 10: '8.3'
Error in sys.excepthook:
Traceback (most recent call last):
File "/usr/lib/python3/dist-packages/apport_python_hook.py", line 63, in apport_excepthook
from apport.fileutils import likely_packaged, get_recent_crashes
File "/usr/lib/python3/dist-packages/apport/__init__.py", line 5, in <module>
from apport.report import Report
File "/usr/lib/python3/dist-packages/apport/report.py", line 30, in <module>
import apport.fileutils
File "/usr/lib/python3/dist-packages/apport/fileutils.py", line 23, in <module>
from apport.packaging_impl import impl as packaging
File "/usr/lib/python3/dist-packages/apport/packaging_impl.py", line 20, in <module>
import apt
File "/usr/lib/python3/dist-packages/apt/__init__.py", line 34, in <module>
apt_pkg.init_config()
SystemError: E:Unable to read /etc/apt/apt.conf.d/ - opendir (13: Permission denied)

Original exception was:
Traceback (most recent call last):
File "/Aprovados.py", line 5, in <module>
R = int(input())
ValueError: invalid literal for int() with base 10: '8.3'
```

Ver saída amigavel

Figura 50 – Exibição de erro sintático da abordagem original no problema 2.

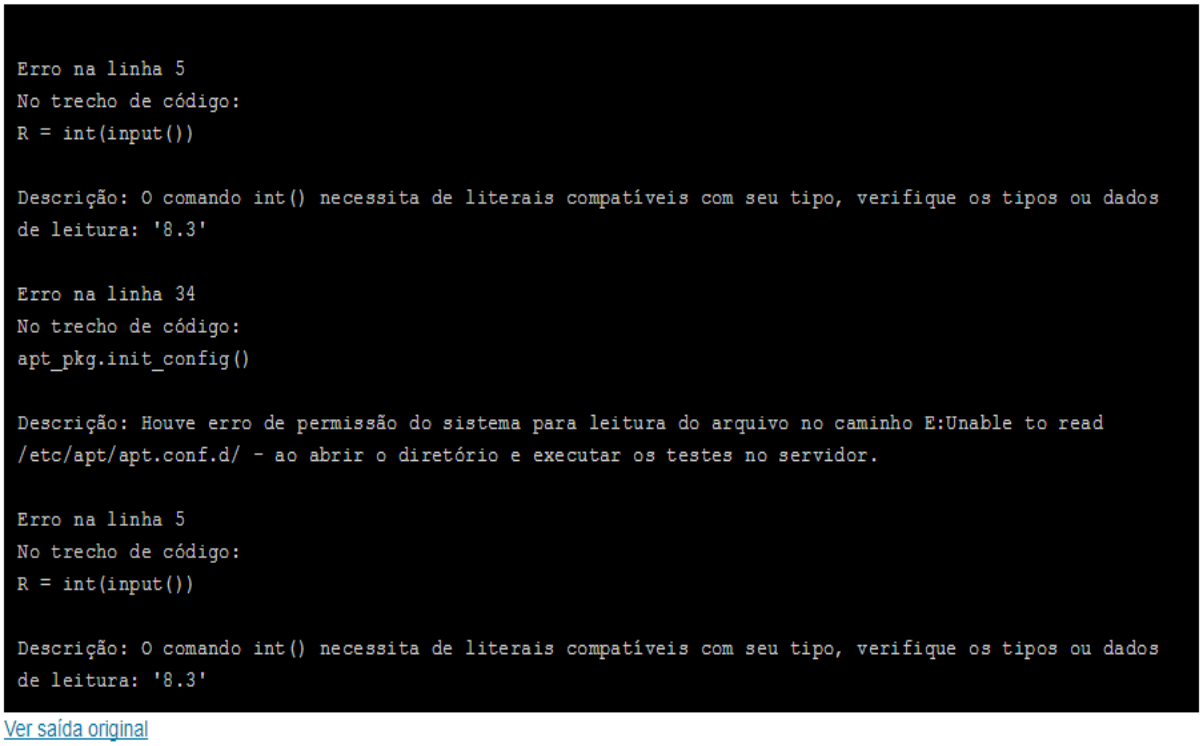

Figura 51 – Exibição de erro sintático da abordagem amigável no problema 2.

Problema 3: Caracteres

As Figuras 52, 54 e 56 ilustram a exibição das mensagens de erros da abordagem original referentes aos erros do problema 3 (três), apresentados durante a execução do experimento. Já as Figuras 53, 55 e 57 ilustram a exibição das mensagens de erros da abordagem amigável referentes aos erros do mesmo problema, apresentados durante a execução do experimento, ao clicar no link "*ver saída amigável*", localizado no canto inferior esquerdo das figuras.

```
File "/Caracteres.py", line 3
letras = input()
^
IndentationError: expected an indented block
Error in sys.excepthook:
Traceback (most recent call last):
File "/usr/lib/python3/dist-packages/apport_python_hook.py", line 63, in apport_excepthook
from apport.fileutils import likely_packaged, get_recent_crashes
File "/usr/lib/python3/dist-packages/apport/__init__.py", line 5, in <module>
from apport.report import Report
File "/usr/lib/python3/dist-packages/apport/report.py", line 30, in <module>
import apport.fileutils
File "/usr/lib/python3/dist-packages/apport/fileutils.py", line 23, in <module>
from apport.packaging_impl import impl as packaging
File "/usr/lib/python3/dist-packages/apport/packaging_impl.py", line 20, in <module>
import apt
File "/usr/lib/python3/dist-packages/apt/__init__.py", line 34, in <module>
apt_pkg.init_config()
SystemError: E:Unable to read /etc/apt/apt.conf.d/ - opendir (13: Permission denied)

Original exception was:
File "/Caracteres.py", line 3
letras = input()
^
IndentationError: expected an indented block
```

Ver saída amigavel

Figura 52 – Exibição de erro sintático da abordagem original no problema 3.

```
Erro na linha 3
No trecho de código:
letras = input()
^

Descrição: Era esperado um bloco indentado, por favor verifique a indentação. A indentação é uma
característica peculiar na linguagem. Enquanto que os blocos são delimitados explicitamente em C, Java e
PHP por chaves e em Pascal e Fortran por palavras-chave como then e endif, em Python blocos são
delimitados por espaços ou tabulações formando uma indentação visual. Não existem símbolos de "abre" e
"fecha". Fique atento para não errar na próxima!

Erro na linha 34
No trecho de código:
apt_pkg.init_config()

Descrição: Houve erro de permissão do sistema para leitura do arquivo no caminho E:Unable to read
/etc/apt/apt.conf.d/ - ao abrir o diretório e executar os testes no servidor.

Erro na linha 3
No trecho de código:
letras = input()
^

Descrição: Era esperado um bloco indentado, por favor verifique a indentação. A indentação é uma
característica peculiar na linguagem. Enquanto que os blocos são delimitados explicitamente em C, Java e
PHP por chaves e em Pascal e Fortran por palavras-chave como then e endif, em Python blocos são
delimitados por espaços ou tabulações formando uma indentação visual. Não existem símbolos de "abre" e
"fecha". Fique atento para não errar na próxima!
```

Ver saída original

Figura 53 – Exibição de erro sintático da abordagem amigável no problema 3.

```
Traceback (most recent call last):
File "/Caracteres.py", line 6, in <module>
print(letras[i], end='')
IndexError: string index out of range
Error in sys.excepthook:
Traceback (most recent call last):
File "/usr/lib/python3/dist-packages/apport_python_hook.py", line 63, in apport_excepthook
from apport.fileutils import likely_packaged, get_recent_crashes
File "/usr/lib/python3/dist-packages/apport/__init__.py", line 5, in <module>
from apport.report import Report
File "/usr/lib/python3/dist-packages/apport/report.py", line 30, in <module>
import apport.fileutils
File "/usr/lib/python3/dist-packages/apport/fileutils.py", line 23, in <module>
from apport.packaging_impl import impl as packaging
File "/usr/lib/python3/dist-packages/apport/packaging_impl.py", line 20, in <module>
import apt
File "/usr/lib/python3/dist-packages/apt/__init__.py", line 34, in <module>
apt_pkg.init_config()
SystemError: E:Unable to read /etc/apt/apt.conf.d/ - opendir (13: Permission denied)

Original exception was:
Traceback (most recent call last):
File "/Caracteres.py", line 6, in <module>
print(letras[i], end='')
IndexError: string index out of range
```

Ver saída amigavel

Figura 54 – Exibição de erro sintático da abordagem original no problema 3.

```
Erro na linha 6
No trecho de código:
print(letras[i], end='')

Descrição: O índice da lista está fora do intervalo, verique o índice de acesso e seu valor.

Erro na linha 34
No trecho de código:
apt_pkg.init_config()

Descrição: Houve erro de permissão do sistema para leitura do arquivo no caminho E:Unable to read
/etc/apt/apt.conf.d/ - ao abrir o diretório e executar os testes no servidor.

Erro na linha 6
No trecho de código:
print(letras[i], end='')

Descrição: O índice da lista está fora do intervalo, verique o índice de acesso e seu valor.
```

Ver saída original

Figura 55 – Exibição de erro sintático da abordagem amigável no problema 3.

```
Traceback (most recent call last):
File "/Caracteres.py", line 7, in <module>
i = i - '1'
TypeError: unsupported operand type(s) for -: 'int' and 'str'
Error in sys.excepthook:
Traceback (most recent call last):
File "/usr/lib/python3/dist-packages/apport_python_hook.py", line 63, in apport_excepthook
from apport.fileutils import likely_packaged, get_recent_crashes
File "/usr/lib/python3/dist-packages/apport/__init__.py", line 5, in <module>
from apport.report import Report
File "/usr/lib/python3/dist-packages/apport/report.py", line 30, in <module>
import apport.fileutils
File "/usr/lib/python3/dist-packages/apport/fileutils.py", line 23, in <module>
from apport.packaging_impl import impl as packaging
File "/usr/lib/python3/dist-packages/apport/packaging_impl.py", line 20, in <module>
import apt
File "/usr/lib/python3/dist-packages/apt/__init__.py", line 34, in <module>
apt_pkg.init_config()
SystemError: E:Unable to read /etc/apt/apt.conf.d/ - opendir (13: Permission denied)

Original exception was:
Traceback (most recent call last):
File "/Caracteres.py", line 7, in <module>
i = i - '1'
TypeError: unsupported operand type(s) for -: 'int' and 'str'
```

Ver saída amigavel

Figura 56 – Exibição de erro sintático da abordagem original no problema 3.

```
Erro na linha 7
No trecho de código:
i = i - '1'

Descrição: A operação - não é suportada para os tipos 'str' e 'int'. Verifique os tipos e suas
operações.

Erro na linha 34
No trecho de código:
apt_pkg.init_config()

Descrição: Houve erro de permissão do sistema para leitura do arquivo no caminho E:Unable to read
/etc/apt/apt.conf.d/ - ao abrir o diretório e executar os testes no servidor.

Erro na linha 7
No trecho de código:
i = i - '1'

Descrição: A operação - não é suportada para os tipos 'str' e 'int'. Verifique os tipos e suas
operações.
```

Ver saída original

Figura 57 – Exibição de erro sintático da abordagem amigável no problema 3.

Problema 4: Desafio do menor e maior número

As Figuras 58, 60 e 62 ilustram a exibição das mensagens de erros da abordagem original referentes aos erros do problema 4 (quatro), apresentados durante a execução do experimento. Já as Figuras 59, 61 e 63 ilustram a exibição das mensagens de erros da abordagem amigável referentes aos erros do mesmo problema, apresentados durante a execução do experimento, ao clicar no link "*ver saída amigável*", localizado no canto inferior esquerdo das figuras.

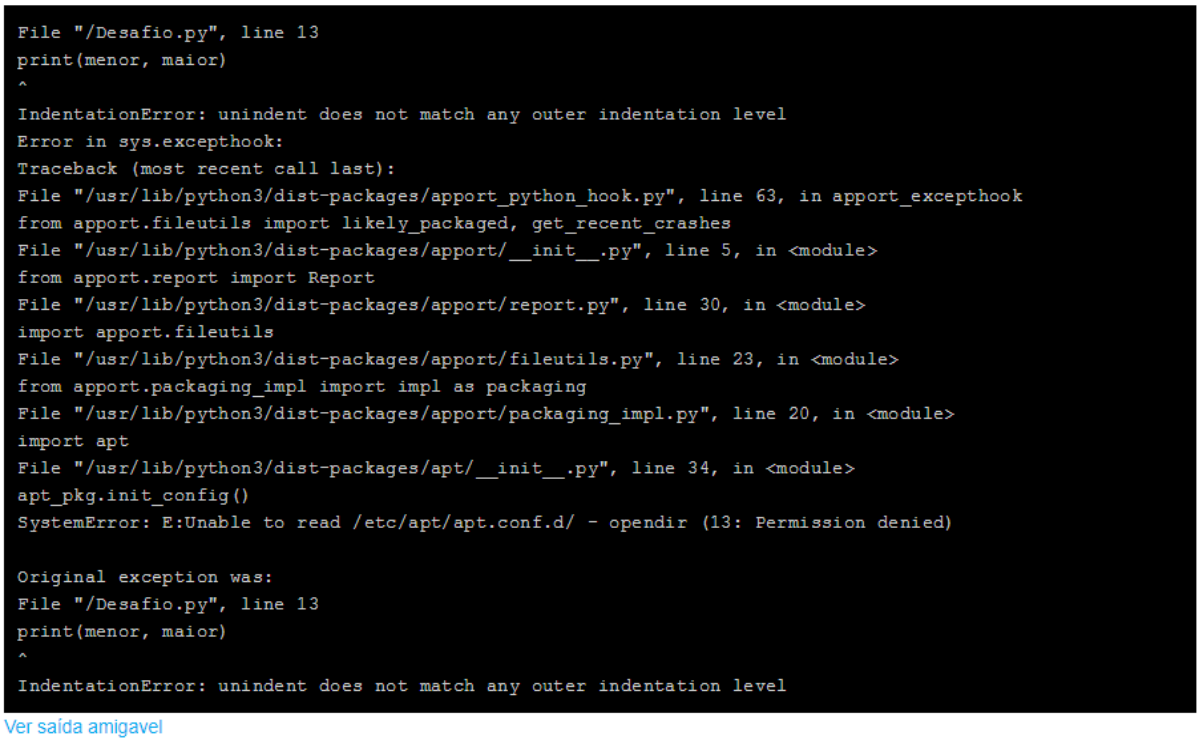

Figura 58 – Exibição de erro sintático da abordagem original no problema 4.

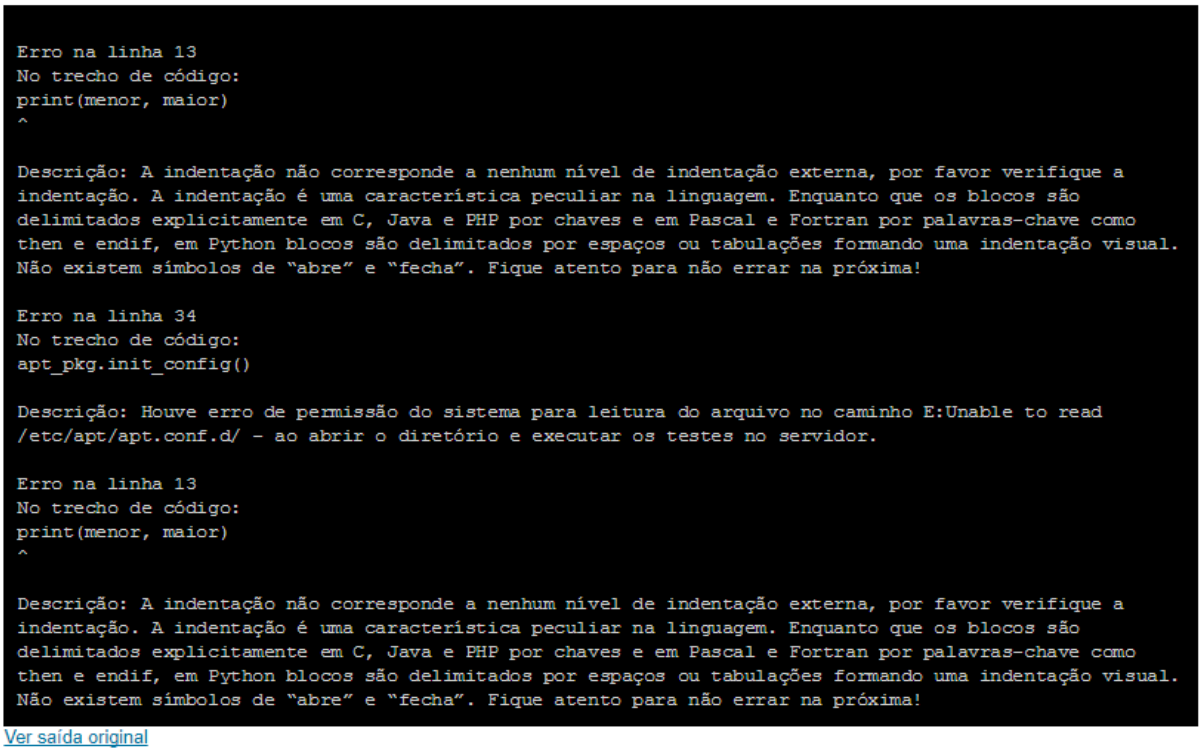

Figura 59 – Exibição de erro sintático da abordagem amigável no problema 4.

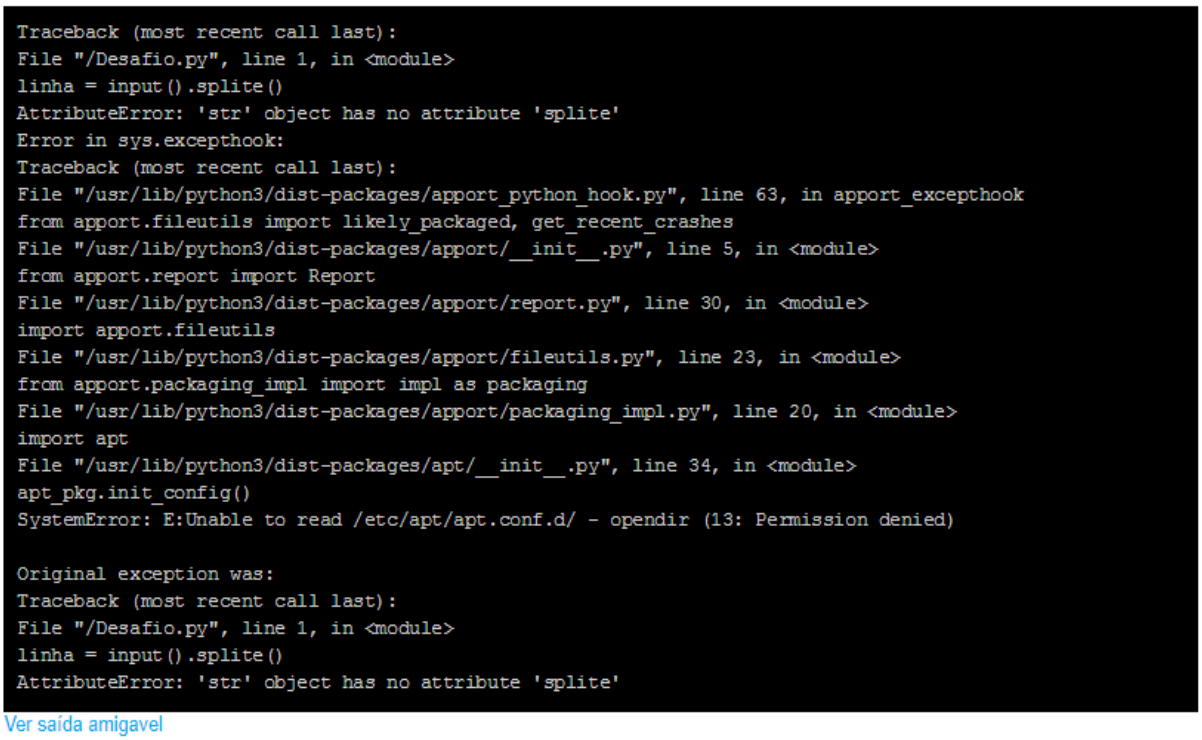

Figura 60 – Exibição de erro sintático da abordagem original no problema 4.

```
Erro na linha 1
No trecho de código:
linha = input().splite()

Descrição: O objeto do tipo 'str' não tem nenhum atributo com o nome 'splite', verifique se digitou
corretamente o nome do atributo ou o objeto.

Erro na linha 34
No trecho de código:
apt_pkg.init_config()

Descrição: Houve erro de permissão do sistema para leitura do arquivo no caminho E:Unable to read
/etc/apt/apt.conf.d/ - ao abrir o diretório e executar os testes no servidor.

Erro na linha 1
No trecho de código:
linha = input().splite()

Descrição: O objeto do tipo 'str' não tem nenhum atributo com o nome 'splite', verifique se digitou
corretamente o nome do atributo ou o objeto.
```

Ver saída original

Figura 61 – Exibição de erro sintático da abordagem amigável no problema 4.

```
Traceback (most recent call last):
File "/Desafio.py", line 7, in <module>
for i in (len(lista)-1):
TypeError: 'int' object is not iterable
Error in sys.excepthook:
Traceback (most recent call last):
File "/usr/lib/python3/dist-packages/apport_python_hook.py", line 63, in apport_excepthook
from apport.fileutils import likely_packaged, get_recent_crashes
File "/usr/lib/python3/dist-packages/apport/__init__.py", line 5, in <module>
from apport.report import Report
File "/usr/lib/python3/dist-packages/apport/report.py", line 30, in <module>
import apport.fileutils
File "/usr/lib/python3/dist-packages/apport/fileutils.py", line 23, in <module>
from apport.packaging_impl import impl as packaging
File "/usr/lib/python3/dist-packages/apport/packaging_impl.py", line 20, in <module>
import apt
File "/usr/lib/python3/dist-packages/apt/__init__.py", line 34, in <module>
apt_pkg.init_config()
SystemError: E:Unable to read /etc/apt/apt.conf.d/ - opendir (13: Permission denied)

Original exception was:
Traceback (most recent call last):
File "/Desafio.py", line 7, in <module>
for i in (len(lista)-1):
TypeError: 'int' object is not iterable
```

Ver saída amigavel

Figura 62 – Exibição de erro sintático da abordagem original no problema 4.

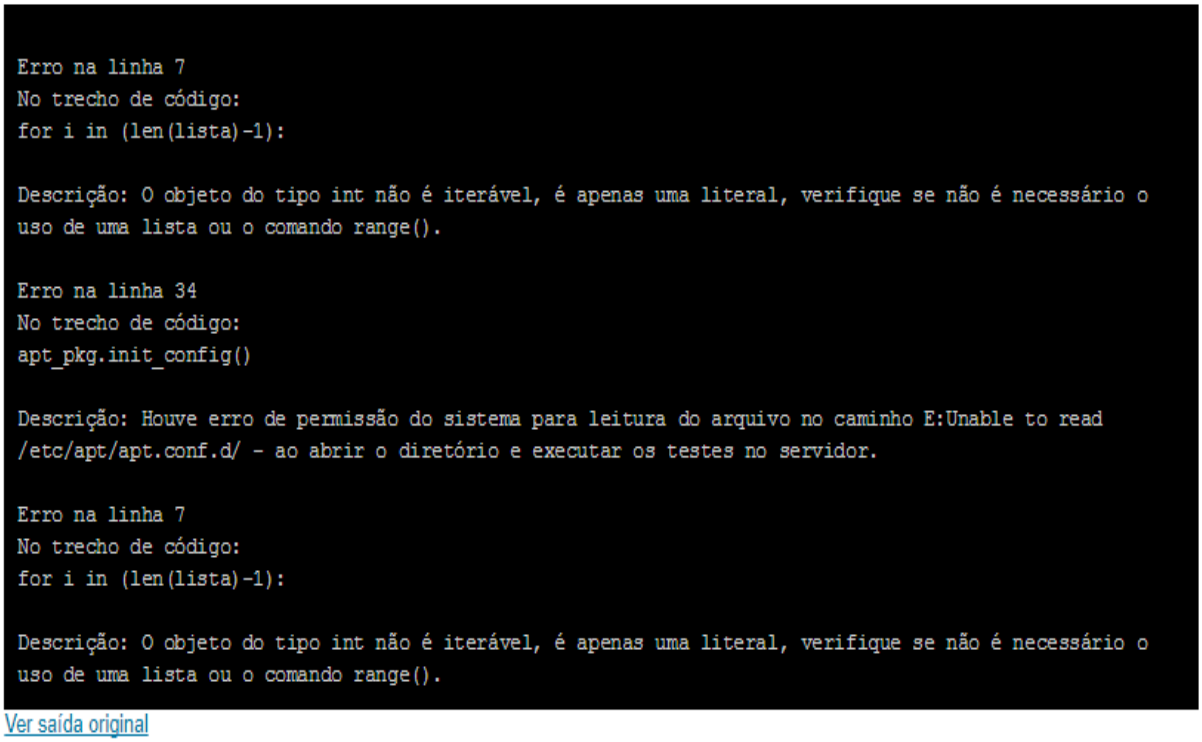

Figura 63 – Exibição de erro sintático da abordagem amigável no problema 4.

Problema 5: Equação do Segundo Grau

As Figuras 64, 66 e 68 ilustram a exibição das mensagens de erros da abordagem original referentes aos erros do problema 5 (cinco), apresentados durante a execução do experimento. Já as Figuras 65, 67 e 69 ilustram a exibição das mensagens de erros da abordagem amigável referentes aos erros do mesmo problema, apresentados durante a execução do experimento, ao clicar no link "*ver saída amigável*", localizado no canto inferior esquerdo das figuras.

```
File "/Equação.py", line 15

^
SyntaxError: unexpected EOF while parsing
Error in sys.excepthook:
Traceback (most recent call last):
File "/usr/lib/python3/dist-packages/apport_python_hook.py", line 63, in apport_excepthook
from apport.fileutils import likely_packaged, get_recent_crashes
File "/usr/lib/python3/dist-packages/apport/__init__.py", line 5, in <module>
from apport.report import Report
File "/usr/lib/python3/dist-packages/apport/report.py", line 30, in <module>
import apport.fileutils
File "/usr/lib/python3/dist-packages/apport/fileutils.py", line 23, in <module>
from apport.packaging_impl import impl as packaging
File "/usr/lib/python3/dist-packages/apport/packaging_impl.py", line 20, in <module>
import apt
File "/usr/lib/python3/dist-packages/apt/__init__.py", line 34, in <module>
apt_pkg.init_config()
SystemError: E:Unable to read /etc/apt/apt.conf.d/ - opendir (13: Permission denied)

Original exception was:
File "/Equação.py", line 15

^
SyntaxError: unexpected EOF while parsing
```

Ver saída amigavel

Figura 64 – Exibição de erro sintático da abordagem original no problema 5.

```
Erro na linha 15
No trecho de código:

^

Descrição: Foram utilizados caracteres que não seguem o padrão da linguagem, assim ocorreu o erro de
final de arquivo inesperado. Verifique os caracteres utilizados.

Erro na linha 34
No trecho de código:
apt_pkg.init_config()

Descrição: Houve erro de permissão do sistema para leitura do arquivo no caminho E:Unable to read
/etc/apt/apt.conf.d/ - ao abrir o diretório e executar os testes no servidor.

Erro na linha 15
No trecho de código:

^

Descrição: Foram utilizados caracteres que não seguem o padrão da linguagem, assim ocorreu o erro de
final de arquivo inesperado. Verifique os caracteres utilizados.
```

Ver saída original

Figura 65 – Exibição de erro sintático da abordagem amigável no problema 5.

```
Traceback (most recent call last):
File "/Equação.py", line 11, in <module>
print("%%.2f"%x1)
TypeError: not all arguments converted during string formatting
Error in sys.excepthook:
Traceback (most recent call last):
File "/usr/lib/python3/dist-packages/apport_python_hook.py", line 63, in apport_excepthook
from apport.fileutils import likely_packaged, get_recent_crashes
File "/usr/lib/python3/dist-packages/apport/__init__.py", line 5, in <module>
from apport.report import Report
File "/usr/lib/python3/dist-packages/apport/report.py", line 30, in <module>
import apport.fileutils
File "/usr/lib/python3/dist-packages/apport/fileutils.py", line 23, in <module>
from apport.packaging_impl import impl as packaging
File "/usr/lib/python3/dist-packages/apport/packaging_impl.py", line 20, in <module>
import apt
File "/usr/lib/python3/dist-packages/apt/__init__.py", line 34, in <module>
apt_pkg.init_config()
SystemError: E:Unable to read /etc/apt/apt.conf.d/ - opendir (13: Permission denied)

Original exception was:
Traceback (most recent call last):
File "/Equação.py", line 11, in <module>
print("%%.2f"%x1)
TypeError: not all arguments converted during string formatting
```

Ver saída amigavel

Figura 66 – Exibição de erro sintático da abordagem original no problema 5.

```
Erro na linha 11
No trecho de código:
print("%%.2f"%x1)

Descrição: Nem todos os argumentos foram convertidos durante a formatação. Verifique a formatação, os
tipos das variáveis e argumentos.

Erro na linha 34
No trecho de código:
apt_pkg.init_config()

Descrição: Houve erro de permissão do sistema para leitura do arquivo no caminho E:Unable to read
/etc/apt/apt.conf.d/ - ao abrir o diretório e executar os testes no servidor.

Erro na linha 11
No trecho de código:
print("%%.2f"%x1)

Descrição: Nem todos os argumentos foram convertidos durante a formatação. Verifique a formatação, os
tipos das variáveis e argumentos.
```

Ver saída original

Figura 67 – Exibição de erro sintático da abordagem amigável no problema 5.

```
Traceback (most recent call last):
File "/Equação.py", line 5, in <module>
x1=((-b)+d**(1/2))/(2*a)
ZeroDivisionError: float division by zero
Error in sys.excepthook:
Traceback (most recent call last):
File "/usr/lib/python3/dist-packages/apport_python_hook.py", line 63, in apport_excepthook
from apport.fileutils import likely_packaged, get_recent_crashes
File "/usr/lib/python3/dist-packages/apport/__init__.py", line 5, in <module>
from apport.report import Report
File "/usr/lib/python3/dist-packages/apport/report.py", line 30, in <module>
import apport.fileutils
File "/usr/lib/python3/dist-packages/apport/fileutils.py", line 23, in <module>
from apport.packaging_impl import impl as packaging
File "/usr/lib/python3/dist-packages/apport/packaging_impl.py", line 20, in <module>
import apt
File "/usr/lib/python3/dist-packages/apt/__init__.py", line 34, in <module>
apt_pkg.init_config()
SystemError: E:Unable to read /etc/apt/apt.conf.d/ - opendir (13: Permission denied)

Original exception was:
Traceback (most recent call last):
File "/Equação.py", line 5, in <module>
x1=((-b)+d**(1/2))/(2*a)
ZeroDivisionError: float division by zero
```

Ver saída amigavel

Figura 68 – Exibição de erro sintático da abordagem original no problema 5.

```
Erro na linha 5
No trecho de código:
x1=((-b)+d**(1/2))/(2*a)

Descrição: Não é possível realizar divisão por zero, verifique se isso ocorre em seu código.

Erro na linha 34
No trecho de código:
apt_pkg.init_config()

Descrição: Houve erro de permissão do sistema para leitura do arquivo no caminho E:Unable to read
/etc/apt/apt.conf.d/ - ao abrir o diretório e executar os testes no servidor.

Erro na linha 5
No trecho de código:
x1=((-b)+d**(1/2))/(2*a)

Descrição: Não é possível realizar divisão por zero, verifique se isso ocorre em seu código.
```

Ver saída original

Figura 69 – Exibição de erro sintático da abordagem amigável no problema 5.

Problema 6: Balanceando números

As Figuras 70, 72 e 74 ilustram a exibição das mensagens de erros da abordagem original referentes aos erros do problema 6 (seis), apresentados durante a execução do experimento. Já as Figuras 71, 73 e 75 ilustram a exibição das mensagens de erros da abordagem amigável referentes aos erros do mesmo problema, apresentados durante a execução do experimento, ao clicar no link "*ver saída amigável*", localizado no canto inferior esquerdo das figuras.

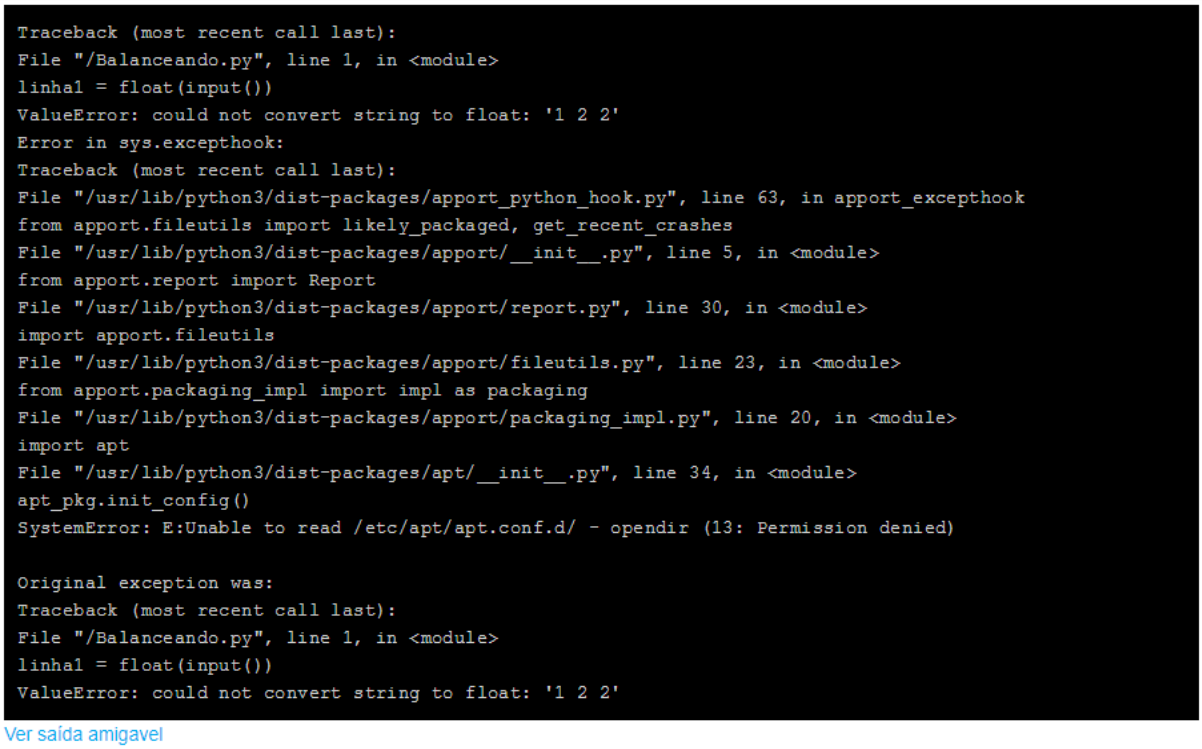

Figura 70 – Exibição de erro sintático da abordagem original no problema 6.

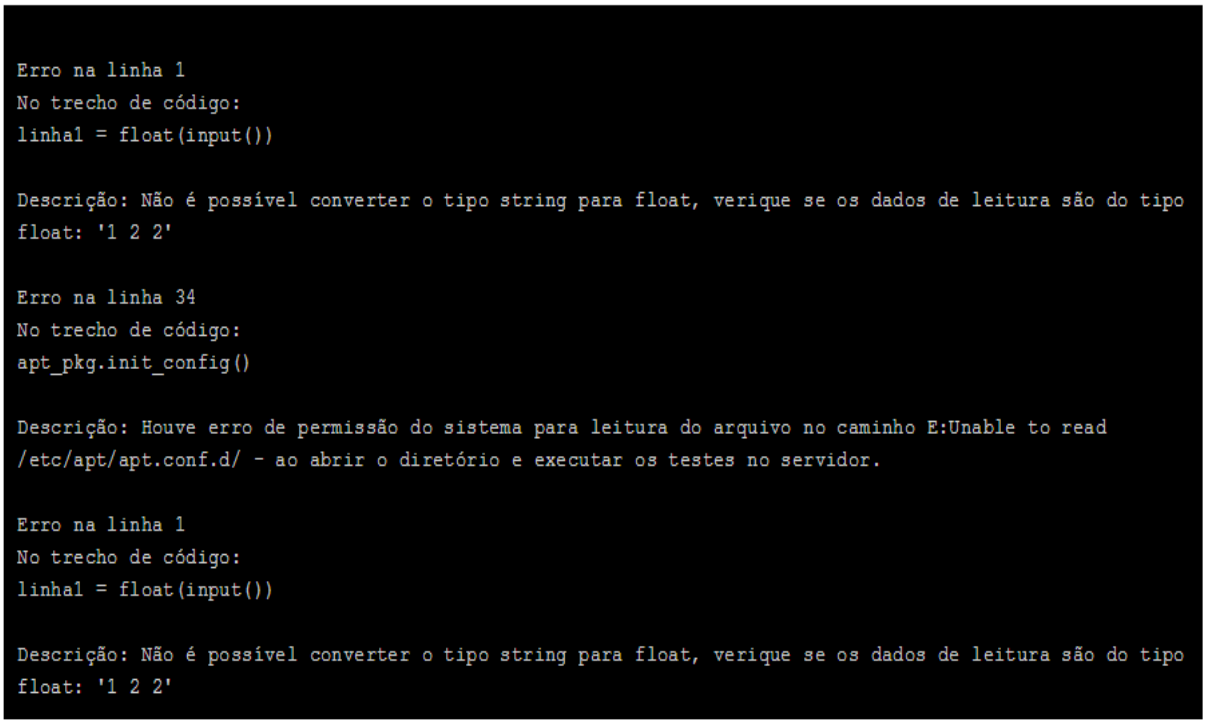

Figura 71 – Exibição de erro sintático da abordagem amigável no problema 6.

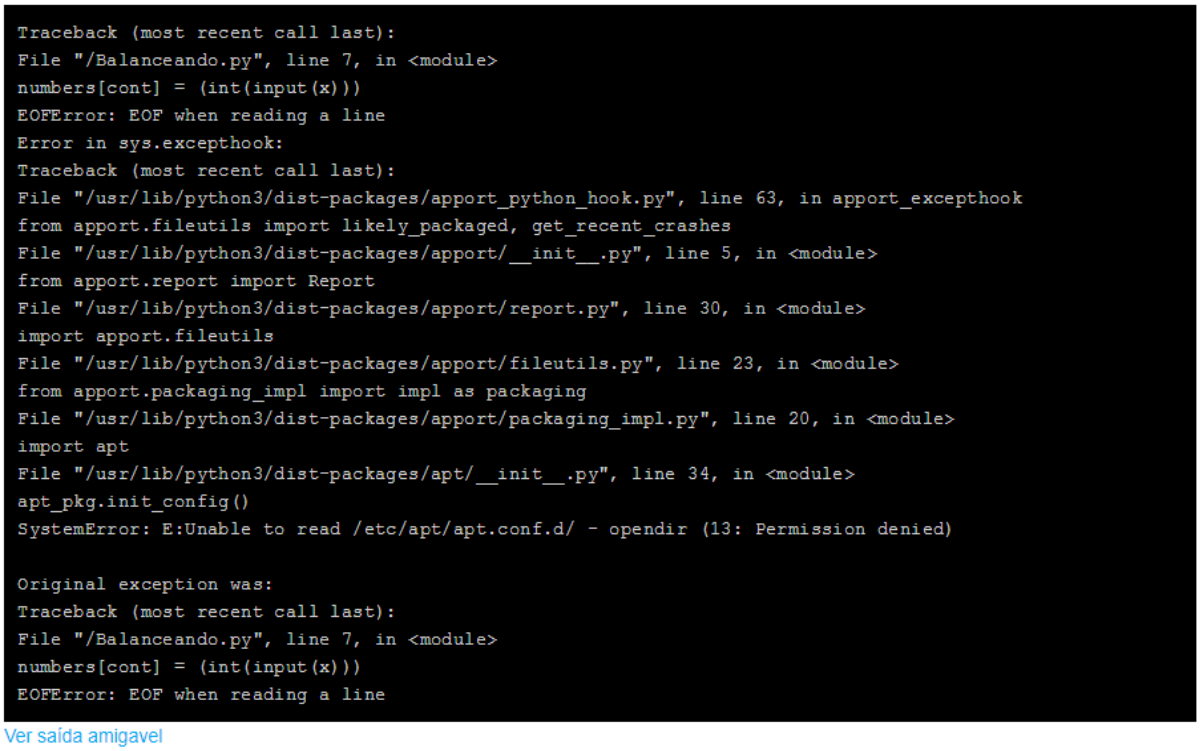

Figura 72 – Exibição de erro sintático da abordagem original no problema 6.

```
Erro na linha 7
No trecho de código:
numbers[cont] = (int(input(x)))

Descrição: EOF - end of file (final de arquivo) - ocorreu um erro de final de arquivo ao realizar a
leitura dos dados, verifique a declaração do input ou as entradas.

Erro na linha 34
No trecho de código:
apt_pkg.init_config()

Descrição: Houve erro de permissão do sistema para leitura do arquivo no caminho E:Unable to read
/etc/apt/apt.conf.d/ - ao abrir o diretório e executar os testes no servidor.

Erro na linha 7
No trecho de código:
numbers[cont] = (int(input(x)))

Descrição: EOF - end of file (final de arquivo) - ocorreu um erro de final de arquivo ao realizar a
leitura dos dados, verifique a declaração do input ou as entradas.
```

Ver saída original

Figura 73 – Exibição de erro sintático da abordagem amigável no problema 6.

```
Traceback (most recent call last):
File "/Balanceando.py", line 12, in <module>
if int(numbers[pos]) == a:
KeyError: 7
Error in sys.excepthook:
Traceback (most recent call last):
File "/usr/lib/python3/dist-packages/apport_python_hook.py", line 63, in apport_excepthook
from apport.fileutils import likely_packaged, get_recent_crashes
File "/usr/lib/python3/dist-packages/apport/__init__.py", line 5, in <module>
from apport.report import Report
File "/usr/lib/python3/dist-packages/apport/report.py", line 30, in <module>
import apport.fileutils
File "/usr/lib/python3/dist-packages/apport/fileutils.py", line 23, in <module>
from apport.packaging_impl import impl as packaging
File "/usr/lib/python3/dist-packages/apport/packaging_impl.py", line 20, in <module>
import apt
File "/usr/lib/python3/dist-packages/apt/__init__.py", line 34, in <module>
apt_pkg.init_config()
SystemError: E:Unable to read /etc/apt/apt.conf.d/ - opendir (13: Permission denied)

Original exception was:
Traceback (most recent call last):
File "/Balanceando.py", line 12, in <module>
if int(numbers[pos]) == a:
KeyError: 7
```

Ver saída amigavel

Figura 74 – Exibição de erro sintático da abordagem original no problema 6.

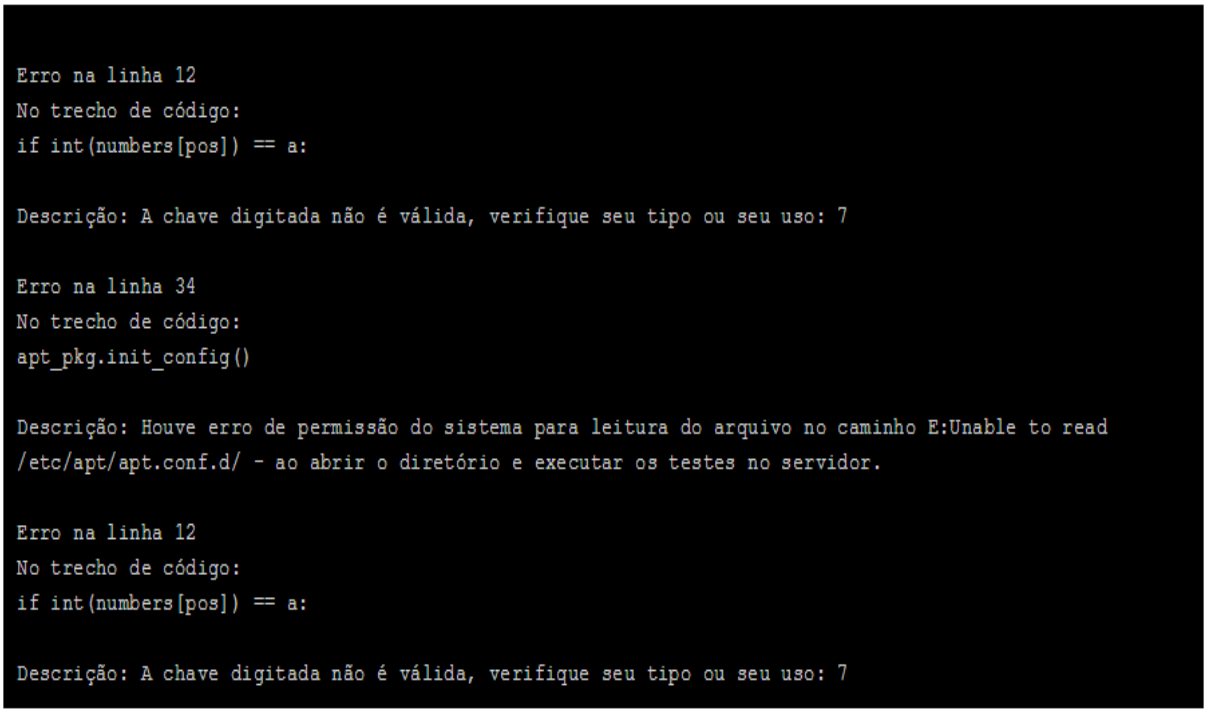

Figura 75 – Exibição de erro sintático da abordagem amigável no problema 6.

APÊNDICE C – Dados do experimento

Neste apêndice são ilustrados os dados colhidos da base do juiz *on-line* The Huxley a partir da realização do experimento, ilustrados em forma de tabela e agrupados por participante, colhendo métricas como as que seguem e ilustradas na Tabela 18:

- Identificação do participante;
- Abordagem utilizada, se foi a mensagem amigável ou original;
- O número de submissões realizadas pelo participante para um determinado problema;
- A quantidade de erros corrigidos;
- O tempo em segundos gasto para resolver um problema;
- O resultado da tentativa de acordo com as submissões, ou seja, se resolveu todos os erros ou não, de acordo com a notação ilustrada na Tabela 7;
- A identificação do problema;
- O nível de inglês do participante;
- Por fim, a média obtida pelo aluno ao final da disciplina cursada.

Tabela 18 – Dados colhidos do experimento.

Participante	Abordagem	Nº Subm.	Corrigidos	Tempo (s)	Resultado	Problema	Nível Inglês	Média
1	Original	4	3	76	FIX	p1	4	9.7
2	Amigável	3	3	138	FIX	p1	4	5.0
3	Original	10	2	519	UFIX	p1	4	5.0
4	Amigável	4	3	134	FIX	p1	2	6.1
5	Amigável	4	3	89	FIX	p1	5	8.4
6	Original	5	3	141	FIX	p1	5	6.1
7	Original	3	3	90	FIX	p1	3	8.7
8	Amigável	5	3	150	FIX	p1	2	5.8
9	Original	16	2	857	UFIX	p1	2	8.1
10	Amigável	4	3	110	FIX	p1	3	6.6
11	Original	9	3	418	FIX	p1	2	5.3
12	Amigável	4	3	165	FIX	p1	2	5.2
13	Amigável	4	3	165	FIX	p1	5	7.6
14	Original	8	3	273	FIX	p1	2	7.8
15	Original	5	3	157	FIX	p1	3	6.3
16	Amigável	7	3	182	FIX	p1	2	5.3
17	Original	4	3	134	FIX	p1	1	5.9
18	Amigável	5	3	187	FIX	p1	3	5.8

Tabela 18 – Dados colhidos do experimento.

Participante	Abordagem	Nº Subm.	Corrigidos	Tempo (s)	Resultado	Problema	Nível Inglês	Média
19	Original	4	3	154	FIX	p1	3	8.3
20	Amigável	11	0	703	UNR	p1	2	2.3
21	Original	14	1	613	UFIX	p1	3	1.8
22	Amigável	5	3	200	FIX	p1	4	5.0
23	Amigável	3	3	137	FIX	p1	2	6.7
24	Original	5	3	198	FIX	p1	4	3.3
25	Original	3	3	166	FIX	p1	4	8.0
26	Amigável	4	3	186	FIX	p1	4	5.3
1	Original	4	3	123	FIX	p2	4	9.7
2	Amigável	3	3	91	FIX	p2	4	5.0
3	Amigável	6	3	239	FIX	p2	4	5.0
4	Original	3	3	113	FIX	p2	2	6.1
5	Original	4	3	86	FIX	p2	5	8.4
6	Amigável	4	3	98	FIX	p2	5	6.1
7	Amigável	4	3	106	FIX	p2	3	8.7
8	Original	5	3	162	FIX	p2	2	5.8
9	Amigável	4	3	218	FIX	p2	2	8.1
10	Original	5	3	158	FIX	p2	3	6.6

Tabela 18 – Dados colhidos do experimento.

Participante	Abordagem	Nº Subm.	Corrigidos	Tempo (s)	Resultado	Problema	Nível Inglês	Média
11	Original	7	3	259	FIX	p2	2	5.3
12	Amigável	6	3	226	FIX	p2	2	5.2
13	Original	5	3	146	FIX	p2	5	7.6
14	Amigável	5	3	145	FIX	p2	2	7.8
15	Original	3	3	44	FIX	p2	3	6.3
16	Amigável	5	3	139	FIX	p2	2	5.3
17	Amigável	6	3	215	FIX	p2	1	5.9
18	Original	5	3	154	FIX	p2	3	5.8
19	Amigável	3	3	96	FIX	p2	3	8.3
20	Original	6	3	310	FIX	p2	2	2.3
21	Amigável	17	2	758	UNR	p2	3	1.8
22	Original	4	3	142	FIX	p2	4	5.0
23	Amigável	4	3	158	FIX	p2	2	6.7
24	Original	3	3	96	FIX	p2	4	3.3
25	Original	3	3	46	FIX	p2	4	8.0
26	Amigável	9	2	608	UNR	p2	4	5.3
1	Amigável	3	3	231	FIX	p3	4	9.7
2	Original	6	2	357	UFIX	p3	4	5.0

Tabela 18 – Dados colhidos do experimento.

Participante	Abordagem	Nº Subm.	Corrigidos	Tempo (s)	Resultado	Problema	Nível Inglês	Média
3	Original	7	2	405	UFIX	p3	4	5.0
4	Amigável	7	3	231	FIX	p3	2	6.1
5	Original	4	3	211	FIX	p3	5	8.4
6	Amigável	7	1	726	UFIX	p3	5	6.1
7	Original	20	2	903	UFIX	p3	3	8.7
8	Amigável	9	2	442	UFIX	p3	2	5.8
9	Original	8	2	526	UFIX	p3	2	8.1
10	Amigável	6	2	394	UFIX	p3	3	6.6
11	Amigável	13	2	655	UFIX	p3	2	5.3
12	Original	7	0	477	UFIX	p3	2	5.2
13	Original	8	1	763	UFIX	p3	5	7.6
14	Amigável	6	3	175	FIX	p3	2	7.8
15	Original	9	1	532	UFIX	p3	3	6.3
16	Amigável	11	3	305	FIX	p3	2	5.3
17	Original	9	3	441	FIX	p3	1	5.9
18	Amigável	9	1	563	UFIX	p3	3	5.8
19	Amigável	9	3	528	FIX	p3	3	8.3
20	Original	11	1	405	UFIX	p3	2	2.3

Tabela 18 – Dados colhidos do experimento.

Participante	Abordagem	Nº Subm.	Corrigidos	Tempo (s)	Resultado	Problema	Nível Inglês	Média
21	Amigável	14	1	553	UFIX	p3	3	1.8
22	Original	20	0	932	UNR	p3	4	5.0
23	Amigável	8	2	539	UFIX	p3	2	6.7
24	Original	5	1	220	UNR	p3	4	3.3
25	Amigável	4	3	234	FIX	p3	4	8.0
26	Original	2	0	40	UFIX	p3	4	5.3
1	Amigável	4	3	142	FIX	p4	4	9.7
2	Original	8	2	405	UFIX	p4	4	5.0
3	Original	4	3	93	FIX	p4	4	5.0
4	Amigável	5	3	121	FIX	p4	2	6.1
5	Amigável	4	3	98	FIX	p4	5	8.4
6	Original	4	3	85	FIX	p4	5	6.1
7	Amigável	2	3	31	FIX	p4	3	8.7
8	Original	7	2	428	UFIX	p4	2	5.8
9	Amigável	4	3	155	FIX	p4	2	8.1
10	Original	5	3	293	FIX	p4	3	6.6
11	Amigável	10	2	634	UFIX	p4	2	5.3
12	Original	8	2	409	UFIX	p4	2	5.2

Tabela 18 – Dados colhidos do experimento.

Participante	Abordagem	Nº Subm.	Corrigidos	Tempo (s)	Resultado	Problema	Nível Inglês	Média
13	Amigável	4	3	228	FIX	p4	5	7.6
14	Original	3	3	128	FIX	p4	2	7.8
15	Amigável	21	2	1071	UFIX	p4	3	6.3
17	Amigável	9	3	581	FIX	p4	1	5.9
16	Original	4	3	133	FIX	p4	2	5.3
18	Original	9	2	465	UFIX	p4	3	5.8
19	Amigável	5	3	341	FIX	p4	3	8.3
20	Original	2	1	39	UFIX	p4	2	2.3
21	Original	12	3	441	FIX	p4	3	1.8
22	Amigável	3	3	71	FIX	p4	4	5.0
23	Original	5	3	315	FIX	p4	2	6.7
24	Amigável	19	1	494	UNR	p4	4	3.3
25	Amigável	3	3	87	FIX	p4	4	8.0
26	Original	6	2	315	UFIX	p4	4	5.3
1	Amigável	9	3	531	FIX	p5	4	9.7
2	Original	6	3	521	FIX	p5	4	5.0
3	Amigável	6	3	551	FIX	p5	4	5.0
4	Original	7	3	454	FIX	p5	2	6.1

Tabela 18 – Dados colhidos do experimento.

Participante	Abordagem	Nº Subm.	Corrigidos	Tempo (s)	Resultado	Problema	Nível Inglês	Média
5	Amigável	4	3	198	FIX	p5	5	8.4
6	Original	7	3	210	FIX	p5	5	6.1
7	Amigável	4	3	151	FIX	p5	3	8.7
8	Original	8	2	379	UFIX	p5	2	5.8
9	Original	4	3	190	FIX	p5	2	8.1
10	Amigável	5	3	180	FIX	p5	3	6.6
11	Original	13	2	694	UFIX	p5	2	5.3
12	Amigável	6	2	296	UFIX	p5	2	5.2
13	Original	4	3	433	FIX	p5	5	7.6
14	Amigável	4	3	238	FIX	p5	2	7.8
15	Amigável	8	3	293	FIX	p5	3	6.3
16	Original	4	3	278	FIX	p5	2	5.3
17	Amigável	5	3	226	FIX	p5	1	5.9
18	Original	8	2	435	UFIX	p5	3	5.8
19	Original	9	2	349	UFIX	p5	3	8.3
20	Amigável	4	2	372	UFIX	p5	2	2.3
21	Amigável	15	2	579	UFIX	p5	3	1.8
22	Original	4	3	174	FIX	p5	4	5.0

Tabela 18 – Dados colhidos do experimento.

Participante	Abordagem	Nº Subm.	Corrigidos	Tempo (s)	Resultado	Problema	Nível Inglês	Média
23	Original	3	3	129	FIX	p5	2	6.7
24	Amigável	8	2	442	UFIX	p5	4	3.3
25	Amigável	7	3	265	FIX	p5	4	8.0
26	Original	5	3	513	FIX	p5	4	5.3
1	Original	5	2	607	UFIX	p6	4	9.7
2	Amigável	7	2	568	UFIX	p6	4	5.0
3	Amigável	12	2	639	UFIX	p6	4	5.0
4	Original	6	2	358	UFIX	p6	2	6.1
5	Original	8	2	608	UFIX	p6	5	8.4
6	Amigável	7	1	481	UFIX	p6	5	6.1
7	Original	31	2	1631	UFIX	p6	3	8.7
8	Amigável	7	2	498	UFIX	p6	2	5.8
9	Amigável	6	3	678	FIX	p6	2	8.1
10	Original	8	2	420	UFIX	p6	3	6.6
11	Amigável	12	1	893	UFIX	p6	2	5.3
12	Original	8	0	520	UFIX	p6	2	5.2
13	Amigável	10	0	634	UFIX	p6	5	7.6
14	Original	12	3	985	FIX	p6	2	7.8

Tabela 18 – Dados colhidos do experimento.

Participante	Abordagem	Nº Subm.	Corrigidos	Tempo (s)	Resultado	Problema	Nível Inglês	Média
15	Amigável	8	1	355	UFIX	p6	3	6.3
16	Original	19	2	952	UFIX	p6	2	5.3
17	Original	8	1	570	UNR	p6	1	5.9
18	Amigável	7	1	546	UFIX	p6	3	5.8
19	Original	5	1	493	UNR	p6	3	8.3
20	Amigável	8	1	432	UFIX	p6	2	2.3
21	Original	13	0	585	UFIX	p6	3	1.8
22	Amigável	7	0	280	UFIX	p6	4	5.0
23	Original	6	1	526	UFIX	p6	2	6.7
24	Amigável	5	1	217	UFIX	p6	4	3.3
25	Original	6	1	784	UNR	p6	4	8.0
26	Amigável	1	0	0	UFIX	p6	4	5.3

APÊNDICE D – Questionário final do experimento

Neste apêndice são ilustradas as questões que fizeram parte do questionário final do experimento, assim como a discussão de suas respostas e resultados obtidos.

O questionário foi composto por 18 (dezoito) perguntas, sendo 13 (treze) perguntas obrigatórias e 5 (cinco) optativas.

Pergunta 1: Qual seu nome?

Esta pergunta tem por objetivo identificar o participante do experimento.

Pergunta 2: Qual sua idade?

Esta pergunta tem por objetivo ter acesso às idades dos participantes e colher dados quantitativos sobre esta métrica. A Figura 76 ilustra estes dados.

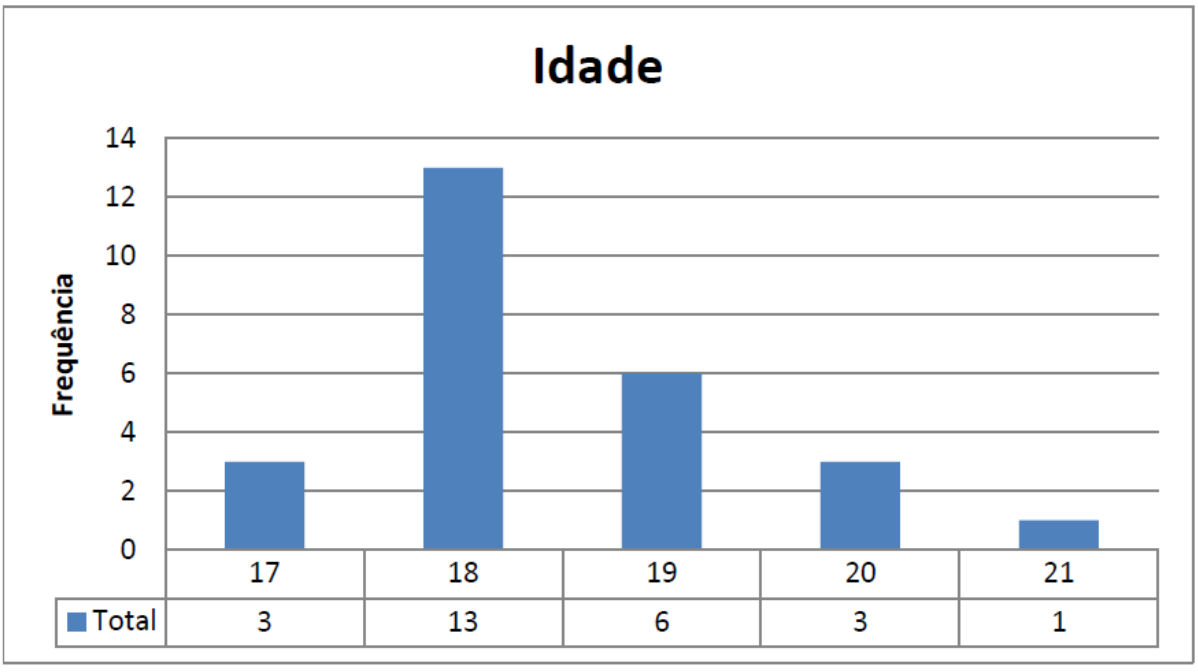

Figura 76 – Gráfico que ilustra as idades dos participantes.

Pergunta 3: Qual seu sexo?

Esta pergunta tem por objetivo ter acesso ao sexo dos participantes e colher dados quantitativos sobre o mesmo. A Figura 77 ilustra estes dados.

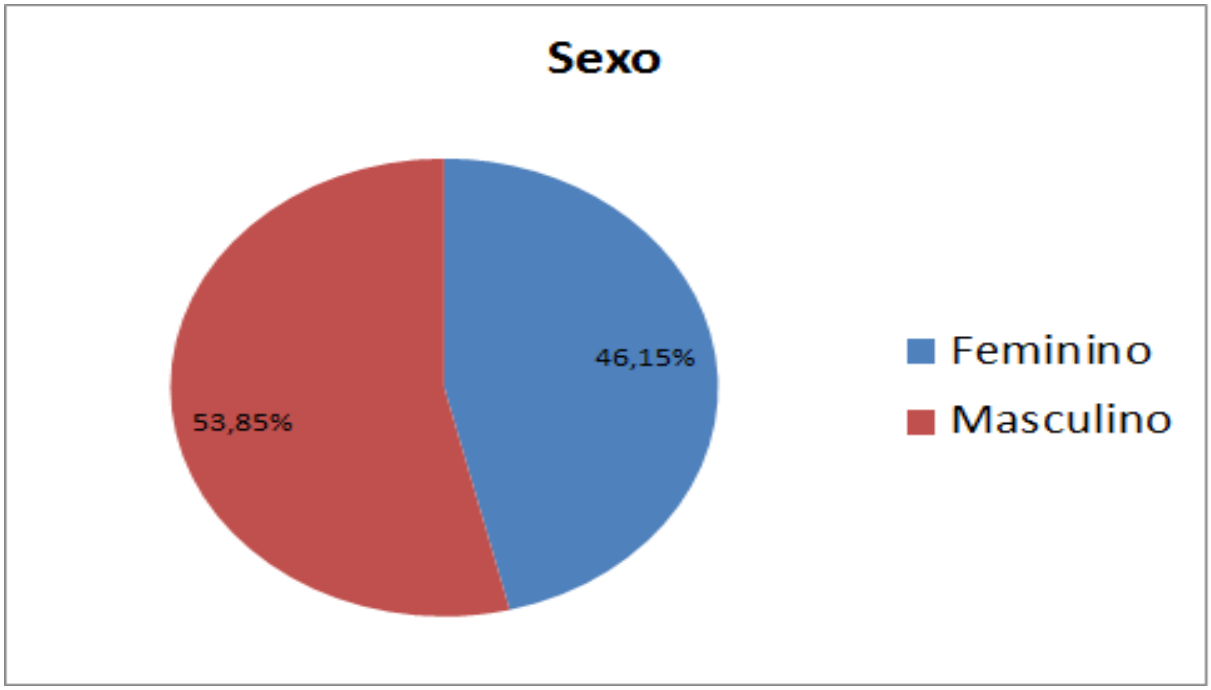

Figura 77 – Gráfico que ilustra o sexo dos participantes.

Pergunta 4: Qual seu curso?

Esta pergunta tem por objetivo ter acesso ao curso dos participantes e colher dados quantitativos sobre o mesmo. A Figura 78 ilustra estes dados.

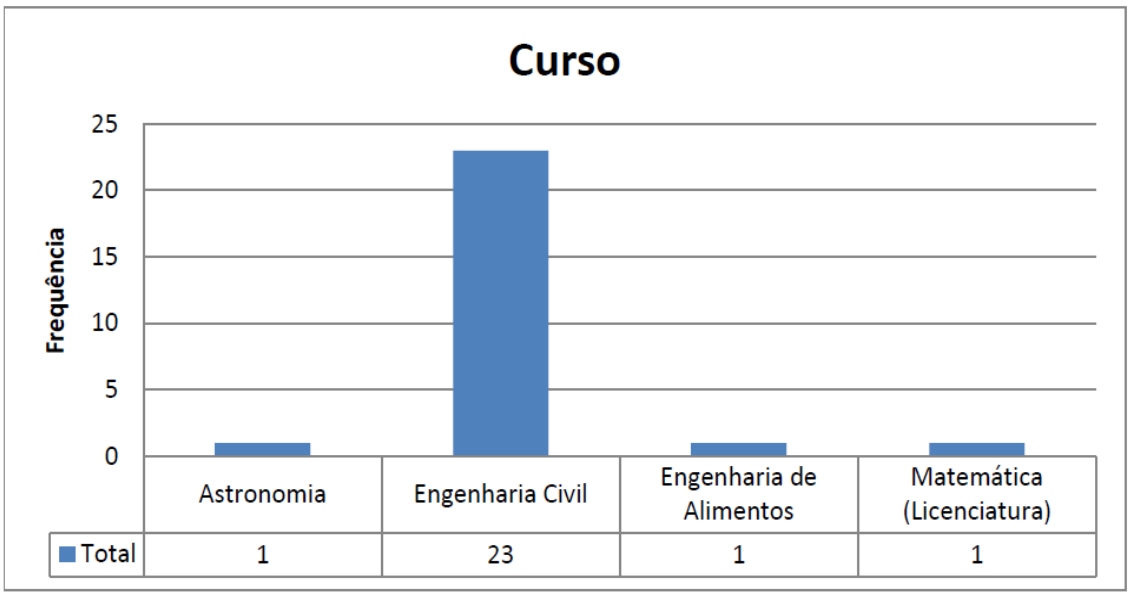

Figura 78 – Gráfico que ilustra o curso dos participantes.

Pergunta 5: Qual o período regular de seu curso?

Esta pergunta tem por objetivo ter acesso ao período regular em que os participantes estão. A Figura 79 ilustra estes dados.

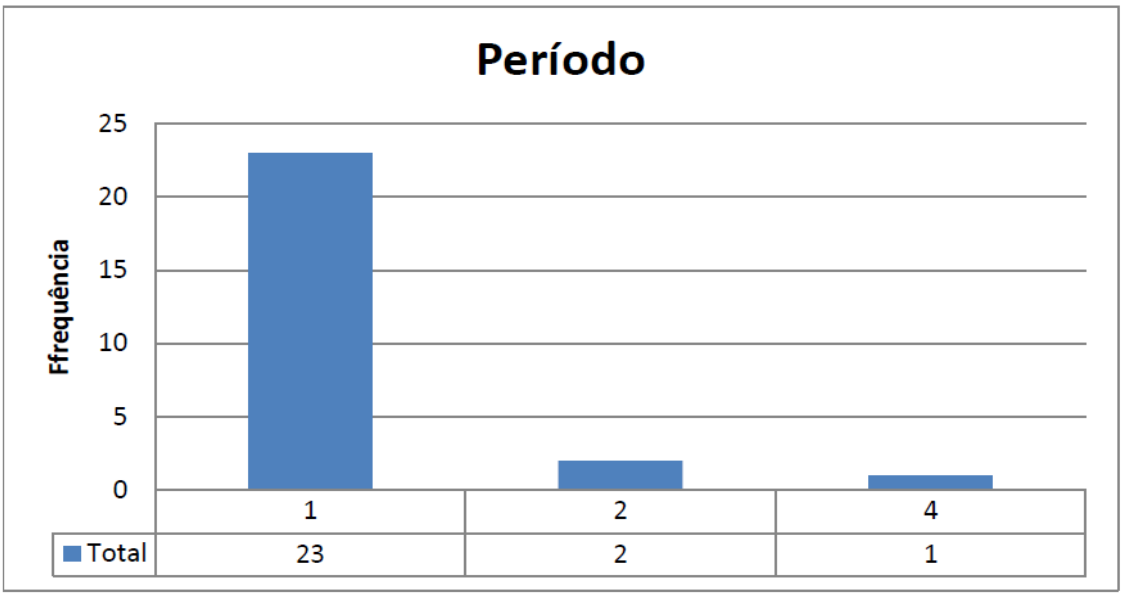

Figura 79 – Gráfico que ilustra o período regular dos participantes.

Pergunta 6: Conseguiu resolver todos os problemas?

Esta pergunta tem por objetivo de saber se o participantes conseguiu resolver todos os problemas. A Figura 80 ilustra estes dados.

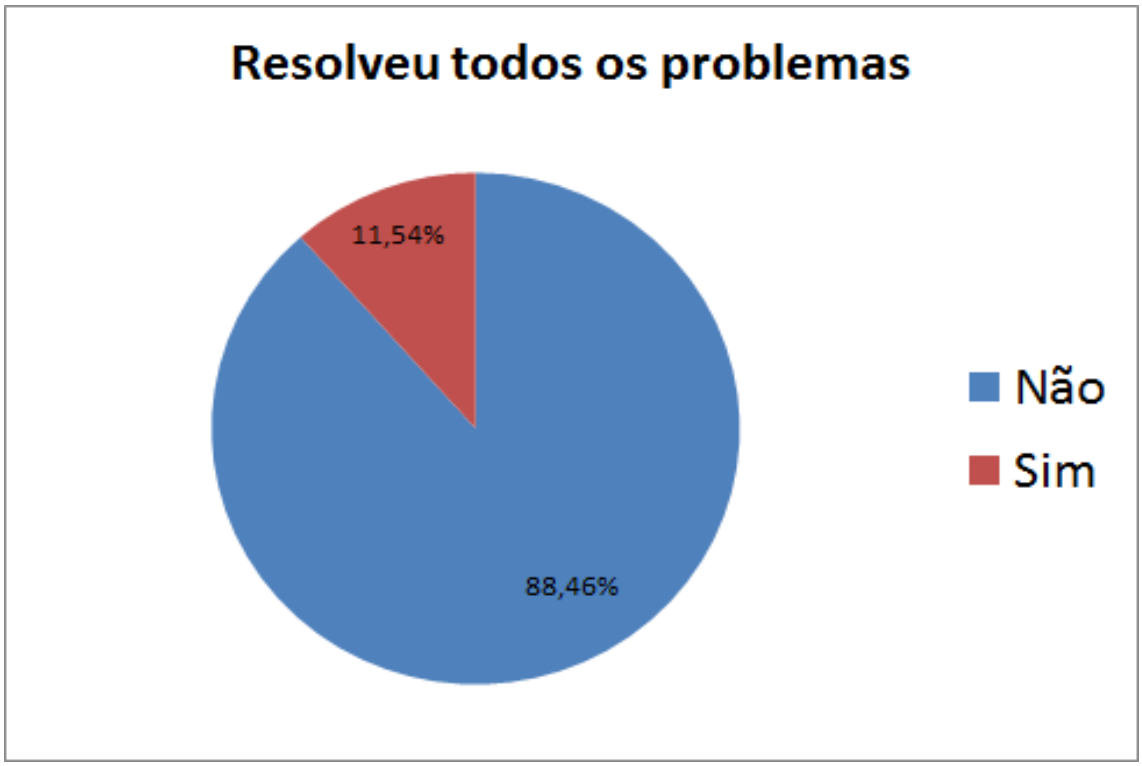

Figura 80 – Gráfico que ilustra o período regular dos participantes.

Pergunta 7: Caso negativo, o que houve e em quais problemas?

Esta é uma pergunta optativa, onde tem o objetivo de saber o motivo pelo qual o participante não conseguiu resolver um determinado problema ou erro, tendo então resposta discursivas, algumas listadas a seguir:

1. Caracteres (não identifiquei o erro do índice) e Balanceando Números;
2. Não conseguir identificar o erro com a ajuda das saídas;
3. Em 10 minutos, não consegui entender a lógica da resposta pronta de um dos problemas. Achei confusa;
4. Não tava entendo o que a saída estava dizendo, envolvia conhecimento sobre dicionários e eu não entendi esse assunto muito bem;
5. Quando não podemos usar a saída amigável fica um pouco mais complicado;
6. Não consegui entender alguns erros / códigos - 3,4,5,6;
7. Não soube identificar/resolver o problema;
8. Passou o tempo, e não entendi o erro;
9. Não consegui identificar o erro de 3 questões;
10. Não consegui reconhecer os erros;
11. Consegui todos exceto "Caracteres"porque ultrapassei os 10 min;
12. Nos dois primeiros tive dificuldade para me concentrar, sendo que no primeiro (problema 6) a mensagem de erro mais atrapalhou do que ajudou;
13. 3, 4 e 6 nao consegui detectar o erro;
14. Não consegui pensar em uma solução em tão pouco tempo que não modificasse tanto o códio original;
15. Infelizmente não tive paciência;
16. Falta de tempo;
17. Tive problemas em entender o programa;
18. Não consegui enxergar o erro à tempo;
19. Resolvi um depois de 10min e balanceamento faltou etapas;
20. Em 5 deles, tenho dificuldade em Programação;

21. Em alguns problemas achei o tempo insuficiente para analisar o programa. Alguns faltam detalhes para finalizar o programa.

Pergunta 8: Já fez alguma outra graduação ou tem experiência anterior com programação?

Esta pergunta tem o objetivo de selecionar os participantes que já tiveram contato anterior com programação, visto que o objetivo do experimento seria aplicar em aprendizes. De acordo com os dados obtidos, nenhum dos participantes tiveram experiência anterior ou já fez algum outro curso de graduação.

Pergunta 9: Caso tenha feito outra graduação, qual?

Esta pergunta tem por objetivo de complementar a anterior, sendo então uma pergunta optativa. Como nenhum dos participantes fez outra graduação ou não tem experiência em programação, nenhum dado foi enviado nesta pergunta.

Pergunta 10: Grau de conhecimento em Python?

Esta pergunta tem o objetivo de obter o grau de conhecimento que os participantes se autoavaliam com relação à linguagem Python, considerando uma escala com cinco opções, seguindo os seguintes critérios: 1-Muito Baixo; 2-Baixo; 3-Médio; 4-Alto; 5-Muito Alto. As respostas obtidas estão ilustradas da Figura 81.

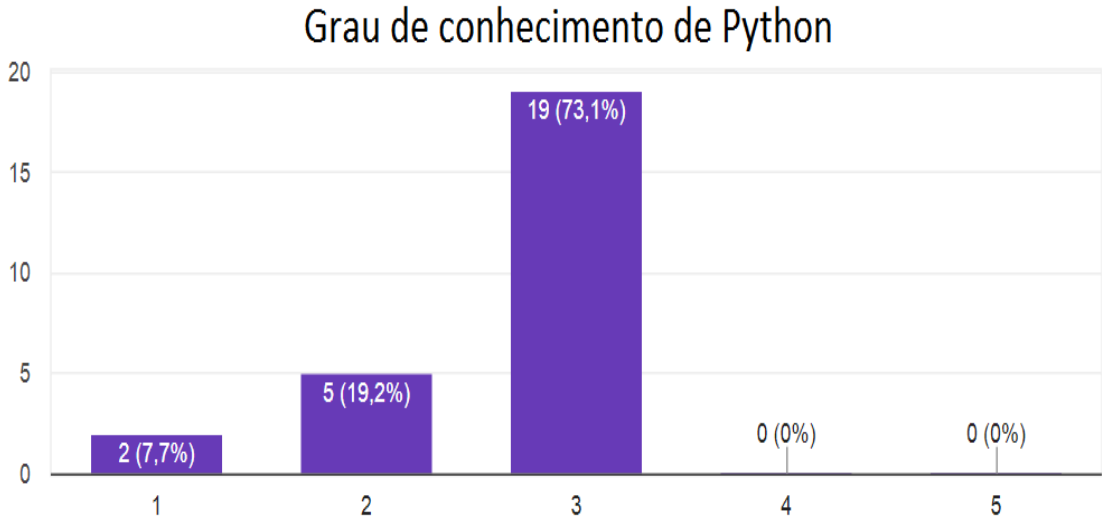

Figura 81 – Gráfico que ilustra a autovaliação de conhecimento da linguagem Python.

Pergunta 11: Qual nível de dificuldade de entender as mensagens de erros originais?

Esta pergunta tem o objetivo de obter o grau de dificuldade que os participantes têm para entender as mensagens apresentadas na abordagem original, considerando uma escala com cinco

opções, seguindo os seguintes critérios: 1-Muito Baixo; 2-Baixo; 3-Médio; 4-Alto; 5-Muito Alto. Os dados referentes às respostas obtidas estão ilustradas da Figura 82.

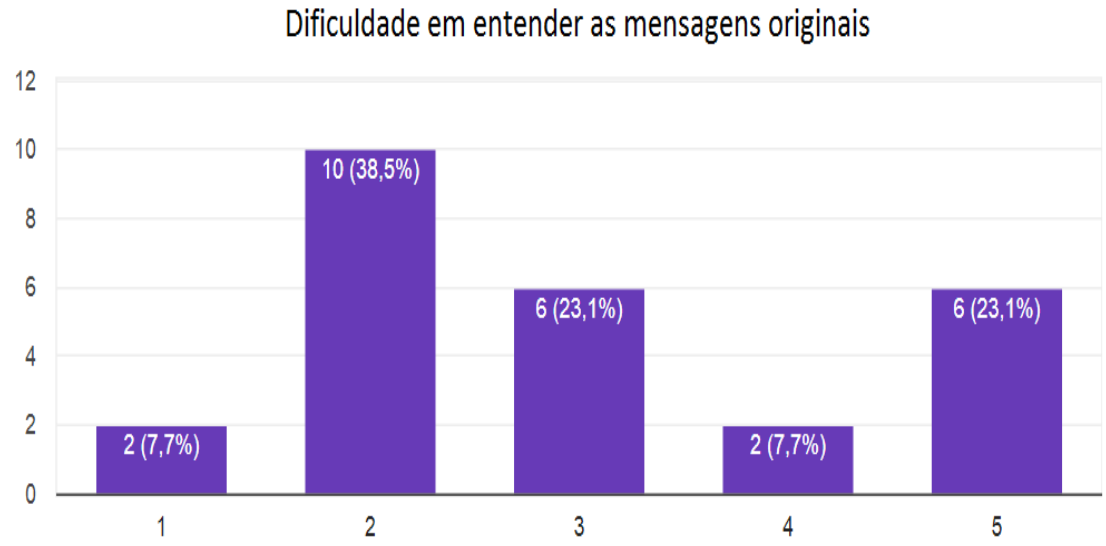

Figura 82 – Gráfico que ilustra o grau de dificuldade em entender as mensagens originais.

Pergunta 12: Qual nível de dificuldade de entender as mensagens amigáveis?

Esta pergunta tem o objetivo de obter o grau de dificuldade que os participantes têm para entender as mensagens apresentadas na abordagem amigável, considerando uma escala com cinco opções, seguindo os seguintes critérios: 1-Muito Baixo; 2-Baixo; 3-Médio; 4-Alto; 5-Muito Alto. Os dados das respostas obtidas estão ilustradas da Figura 83.

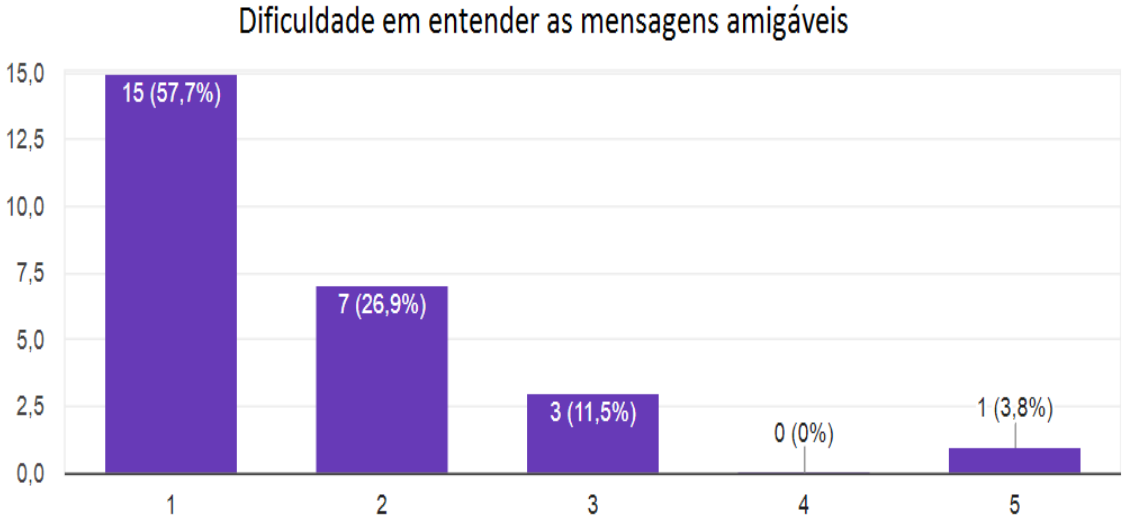

Figura 83 – Gráfico que ilustra o grau de dificuldade em entender as mensagens amigáveis.

Pergunta 13: Você acha importante possuir mensagens amigáveis?

Esta pergunta tem o objetivo saber se o participante acha importante a existência de mensagens amigáveis. De acordo com as respostas, todos os participantes acham que é importante sua existência.

Pergunta 14: Caso ache importante possuir mensagens, explique.

Esta pergunta tem como objetivo complementar a perguntar anterior, colhendo respostas qualitativas acerca das mensagens amigáveis. Algumas destas estão listadas a seguir:

1. Para quem nao entende ingles pode ser importante;
2. Facilita bastante ao aluno identificar o seu erro assim podendo corrigir nas proximas vezes;
3. É importante que ao menos existam mensagens traduzidas para quem não consegue ler as mensagens originais em inglês, mas o direcionamento para os possíveis erros cometidos é sem dúvida muito favorável para quem está aprendendo a programar;
4. Para dar dicas de onde estou errando;
5. Já que eu não entendo muito bem inglês, pra mim é essencial;
6. A percepção do erro fica mais direta;
7. Indica o erro com um precisão maior e direciona melhor para correção;
8. Sim, é um jeito mais claro e limpo de explicar;
9. As mensagens amigaveis traduzem boa parte dos erros que o The Huxley executa, ou seja, torna muito mais facil o entendimento do aluno sobre o que ele está errando;
10. Facilidade em achar erros;
11. É melhor quando é na sua língua;
12. Pra quem não sabe traduzir é uma boa;
13. As mensagens amigáveis conseguem aproximar a linguagem para melhor entendimento do usuário;
14. Considero importante para reportar problemas de semântica além dos de lógica já reportados pela saída original;
15. Com as mensagens amigáveis podemos entender melhor onde erramos e qual o nosso erro, o que ajuda muito na hora de corrigir;
16. Nos ajuda bastante;

17. Facilita a compreensão para aqueles que stão começando;
18. Muito mais simples;
19. Caso não saiba inglês ou não entenda o que a original diz;
20. A forma de explicar ajuda no entendimento;
21. Por não haver tanto conhecimento sobre a linguagem utilizada pelo Huxley para correção, a mensagem amigável é uma ótima saída para entendimento rápido e completo;
22. Esclarece o erro;
23. É importante pois às vezes nos ajudar mais a entender o programa e os futuros programas;
24. O uso da saida amigável deixa mais transparente o problema para que a pessoa possa responder, além de que nem todos tem conhecimento de inglês;
25. Para o estímulo do aluno em resolver as questões sem desistir na metade ou não compreender direito o erro ocorrido.

Pergunta 15: Com que frequência você usa este recurso no The Huxley?

Esta pergunta tem o objetivo saber se o participante usa o mecanismo de mensagens amigáveis ao usar o The Huxley em seu cotidiano. Considerando uma escala com cinco opções, seguindo os seguintes critérios: 1-Muito Baixo; 2-Baixo; 3-Médio; 4-Alto; 5-Muito Alto. Os dados obtidas estão ilustradas da Figura 84.

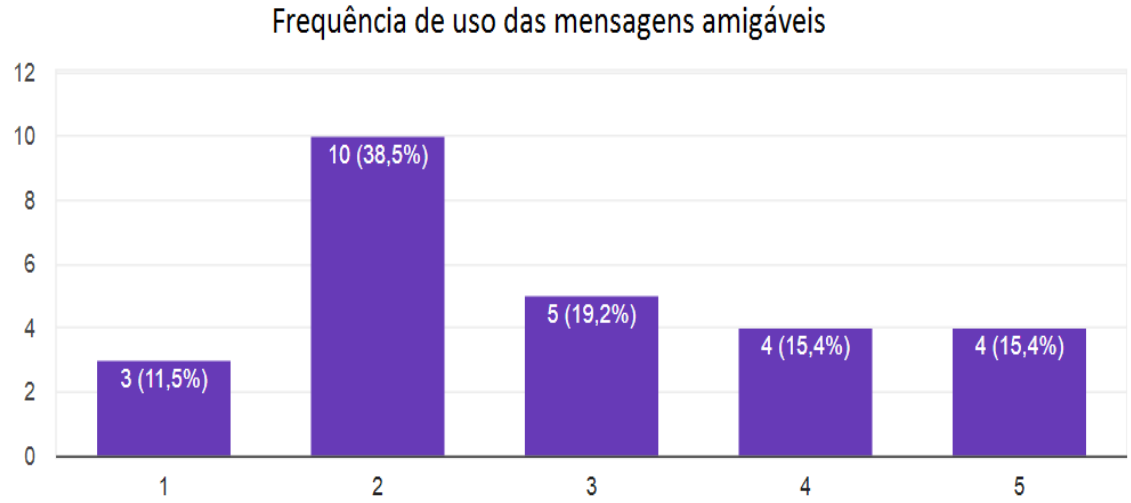

Figura 84 – Gráfico que ilustra o uso de mensagens amigáveis.

Pergunta 16: Caso tenha usado, foi útil?

Esta pergunta tem o objetivo de complementar a perguntar anterior, onde ao participante usar o mecanismo de mensagens amigáveis verificar se o mesmo foi útil. Considerando uma

escala com cinco opções, seguindo os seguintes critérios: 1-Muito Baixo; 2-Baixo; 3-Médio; 4-Alto; 5-Muito Alto. Os dados obtidas estão ilustradas da Figura 85.

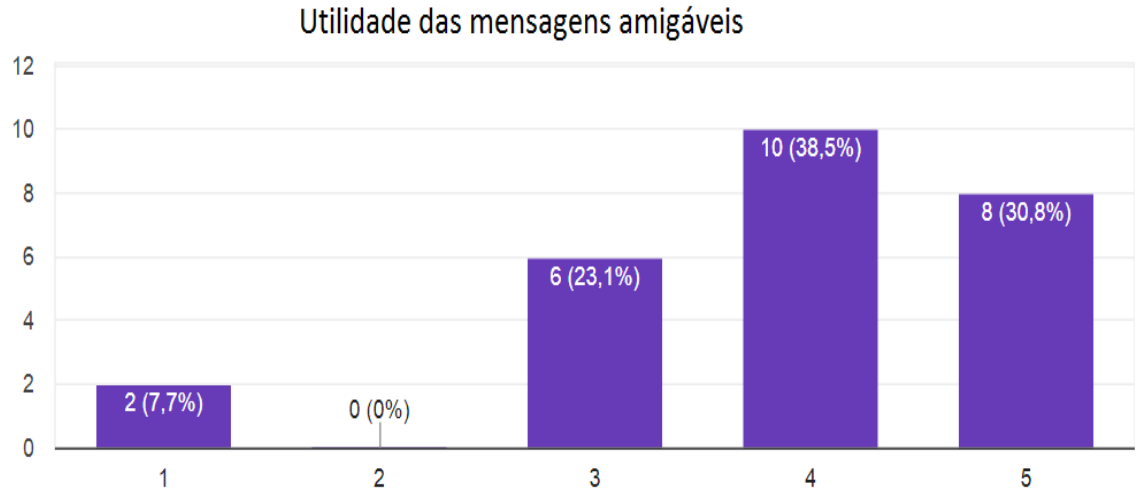

Figura 85 – Gráfico que ilustra a utilidade do uso de mensagens amigáveis.

Pergunta 17: Qual seu nível de conhecimento em inglês?

Esta pergunta tem o objetivo de mensurar o nível de inglês dos participantes, considerando uma escala com cinco opções, seguindo os seguintes critérios: 1-Muito Baixo; 2-Baixo; 3-Médio; 4-Alto; 5-Muito Alto. Sendo útil para análises estatísticas e agrupamento. Os dados obtidos estão ilustrados na Figura 86.

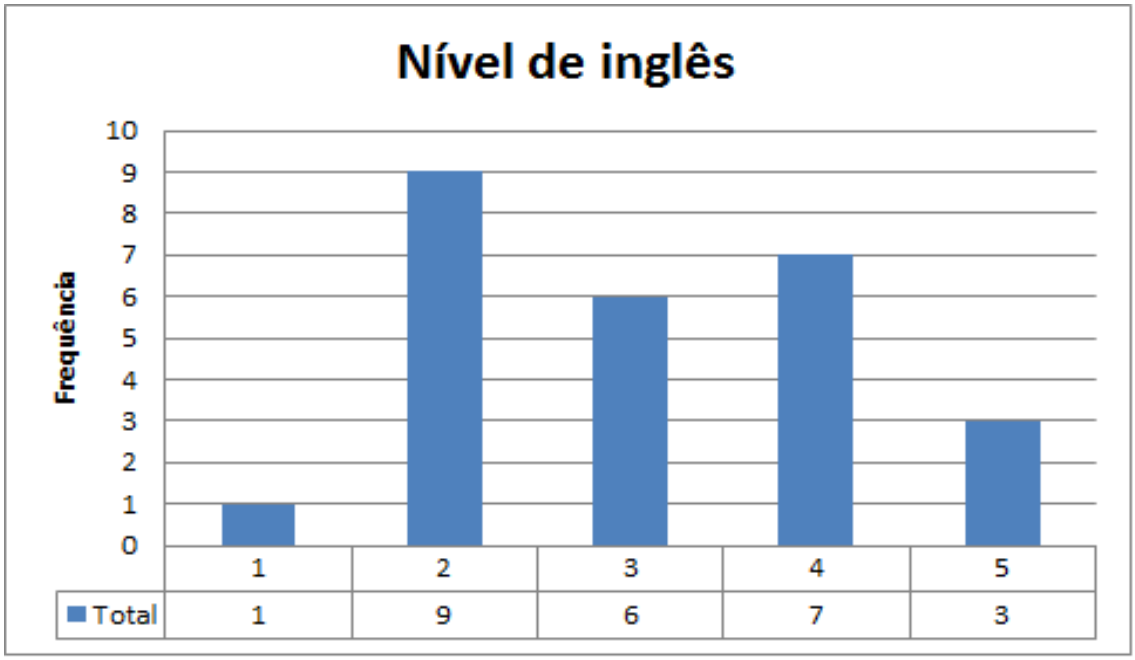

Figura 86 – Gráfico que ilustra o nível de inglês dos participantes.

Pergunta 18: Comentários ou sugestões?

Esta pergunta tem por objetivo colher informações de opiniões dos participantes acerca de comentário ou sugestões, sendo dados qualitativos ou descritivos. Conforme ilustrado a seguir:

1. Algumas das mensagens ajudaram bastante, mas achei desnecessária a explicação detalhada sobre identação (com exemplos de outras linguagens!) para o que deveria ser apenas uma mensagem de erro. Apenas usei o recurso nesse experimento, mas pelo que vi, elas parecem fazer seu trabalho de direcionar o programador aos erros mais prováveis a terem sido cometidos - esse deve ser o foco. Acho que elas devem se manter simples e objetivas;

2. Seria interessante a implementação dos comentários amigáveis na parte da execução também;

3. Da próxima vez que realizar um experimento desse tipo ou qualquer outro, preparar com antecedência os laboratórios, pois se perdeu muito tempo na alocação dos estudantes;

4. Deveria ter mais tempo. E o grau de dificuldade deveria ser aumentado aos poucos. O professor deveria acompanhar de maneira mais eficiente;

5. Ter algum exemplo no The Huxley de como resolvê-los;

6. No caso das respostas , as vezes não existem dicas para ajudar... o ideal é que em todos os programas haja dicas para facilitar o entendimento do aluno.